CodeWarrior®

MPW Cmd-Line Tools Reference

Not to be updated anymore.
See CW Cmd-Line Tools Ref for embedded cmd-line tools info. -- LFT 980825

Because of last-minute changes to CodeWarrior, some of the information in this manual may be inaccurate. Please read the Release Notes on the CodeWarrior CD for the latest up-to-date information.

Revised: 980822 cjm

Metrowerks CodeWarrior copyright ©1993–1998 by Metrowerks Inc. and its licensors. All rights reserved.

Documentation stored on the compact disk(s) may be printed by licensee for personal use. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from Metrowerks Inc.

Metrowerks, the Metrowerks logo, CodeWarrior, and Software at Work are registered trademarks of Metrowerks Inc. PowerPlant and PowerPlant Constructor are trademarks of Metrowerks Inc.

All other trademarks and registered trademarks are the property of their respective owners.

ALL SOFTWARE AND DOCUMENTATION ON THE COMPACT DISK(S) ARE SUBJECT TO THE LICENSE AGREEMENT IN THE CD BOOKLET.

How to Contact Metrowerks:

U.S.A. and international	Metrowerks Corporation 9801 Metric, Suite #100 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
Ordering	Voice: (800) 377–5416 Fax: (512) 873–4901
World Wide Web	`http://www.metrowerks.com`
Registration information	`register@metrowerks.com`
Technical support	`support@metrowerks.com`
Sales, marketing, & licensing	`sales@metrowerks.com`
CompuServe	goto `Metrowerks`

Table of Contents

1 Introduction 17

Read the Release Notes 17
What's New in Command-Line Tools 17
Overview of Command-Line Tools 18
Starting Points 19

2 Getting Started 21

About Command-Line Compilers and Linkers 21
Installation for MPW 22
Installing MPW and the CodeWarrior Tools for MPW 22

3 Programming with MPW 25

MPW Basics 25
The MPW Shell 25
Setting environment variables 26
Executing commands in MPW 27
Using CodeWarrior MPW tools 27
Preparing to use the CodeWarrior tools for MPW 27
Getting Help 27
Using the Tools 28
Include file searching 29
Example programs 29
Speeding Up MPW Builds 29

4 Segmenting 68K Code 33

Overview of Segmenting 68K Code 33
Example Code 33
Segmenting with MPW and MWLink68K 35
Renaming Segments 37
Renaming Several Segments 39
Renaming the First Segment 42
Creating a Segment With No Name 42
A Naming Frenzy 43
Segmenting Special Cases 44
Applications 44

Libraries 44
MPW Tools 44
Code Resources 45
Multi-Segment Drivers and Desk Accessories 49
Changing Resource Attributes 52
Using -ra 52
Setting Attributes for Code Resources 53

5 Metrowerks C/C++ 55

Overview of Metrowerks C/C++ 55
Code Generation Control 56
-a4 56
-[no]a6 56
-align array[members] 57
-align mac68k/mac68k4byte/power/packed 57
-b 58
-b2 58
-Cpp_exceptions on/off 59
-d0 59
-directdestruction 59
-dsize 64/80 60
-ext extension 60
-enum int/min 61
-fatext 61
-fdt num 61
-force_active 62
-fp_contract on/off 62
-fp_library on/off 62
-fp_pilot_traps on/off 63
-inline on/all/off/none/static/auto 63
-intsize 2/4 63
-indirect 64
-ldsize 64 64
-make 64
-mc68020 64

-mc68881 . . . 65
-mpwc . . . 65
-model CFMflat[rf]/CFMflatdf . . . 65
-model farCode/nearCode/codeSmart . . . 65
-model farData/nearData . . . 66
-model farStrings/nearStrings . . . 66
-model farVTables/nearVTables . . . 67
-model far/near . . . 67
-nodirectdestruction . . . 68
-o name . . . 68
-opt *option*[, ...] . . . 68
-O . . . 71
-rostr . . . 71
-RTTI on/off . . . 72
-s name . . . 72
-som . . . 73
-som_env_check . . . 73
-str[ings] dontreuse . . . 73
-str[ings] pool . . . 73
-str ro . . . 74
-str[ings] readonly . . . 74
-toc_data on/off . . . 74
Preprocessing and Precompiling . . . 75
-check precomp[flags] . . . 75
-d name [= [value]] . . . 75
-D name [= [value]] . . . 76
-dump[c] filename . . . 76
-e . . . 76
-load filename . . . 77
-[no]mf . . . 77
-[not]once . . . 77
-pragma somepragma . . . 77
-precompile filename . . . 78
-prefix filename . . . 78
-u name . . . 79

-U name . . . 79
Language Control . . . 79
-ansi off/on/relaxed/strict . . . 79
-appleext on/off . . . 80
-ARM on/off . . . 80
-bool on/off . . . 81
-c. . . . 81
-char signed/unsigned . . . 82
-dialect c/cplus . . . 82
-ec++ on|off . . . 82
-EOL . . . 83
-j0 . . . 83
-j1 . . . 83
-j2 . . . 83
-[no]mapcr . . . 83
-mpwc . . . 84
-objc on|off . . . 84
-relax_pointers . . . 84
-msext on/off . . . 85
-proto auto/strict/ignoreoldstyle . . . 85
-r[equireprotos] . . . 85
-RTTI on/off . . . 85
-strict on/off . . . 86
-trigraphs on/off . . . 86
-wchar_t on/off . . . 86
Input File Control . . . 86
-c. . . . 88
-cwd . . . 88
-i directory . . . 89
-i- . . . 89
-nodefaults and -nostdinc . . . 91
-nosyspath . . . 91
Diagnostic Message Control . . . 92
-help . . . 92
-jm . . . 92

-maxerrors number 92
-make. 92
-nofail 92
-p 93
-t. 93
-version 93
-woption 93
-w[arnings] option[,option...] 93
-w [no]implicitconv. 96
-w [no]notinlined. 96
Debugging Control 97
-g 97
-mbg on/off/full/ch8. 97
-sym on/off/full/fullpath 97
-sym traceback 98
-traceback 98
Profiling Control 98
-profile on/off 98

6 Metrowerks Pascal 101

Overview of Metrowerks Pascal 101
Code Generation Control 102
-a4 102
-[no]a6 103
-align mac68k/mac68k4byte/power 103
-b 104
-b2 104
-d0 104
-dsize 64/80 104
-ext extension 105
-fatext 105
-force_active 105
-fp_contract on/off 106
-ldsize 64 106
-mc68020 106

-mc68881 . . . 106
-model farCode/nearCode/codeSmart . . . 107
-model farData/nearData . . . 107
-model far/near . . . 107
-model farStrings/nearStrings . . . 108
-o name . . . 108
-opt *option*[, ...] . . . 108
-ov . . . 110
-r . . . 110
-str[ings] dontreuse . . . 111
Preprocessing and Precompiling . . . 111
-d name [= [value]] . . . 111
Language Control . . . 111
-ans . . . 111
-c . . . 111
-cs . . . 112
-[no]mf . . . 112
-nilchk . . . 112
-param . . . 112
-relax . . . 112
-sets pool/dontreuse . . . 113
-str[ings] pool/dontreuse/dynamic . . . 113
-trapcase . . . 113
-turbo . . . 113
Input File Control . . . 114
-cwd . . . 115
-i directory . . . 115
-i- . . . 115
Diagnostic Message Control . . . 117
-e filename . . . 117
-h . . . 117
-maxerrors number . . . 117
-p . . . 117
-rebuild . . . 117
-t . . . 117

-w[arnings] option[,option...] 117
Debugging Control 118
-mbg off/full/ch8/on. 118
-clean 119
-k directory 119
-noload 119
-sym on/off/full/fullpath 119
-sym traceback 119
Profiling Control 120
-profile on/off 120

7 Metrowerks Linkers 121

Overview of Metrowerks Linkers 121
Object Code Control 122
-collapsereloads 122
-custom 123
-dead on/off/code/initcode 123
-intsize 2/4 123
-model near/far 124
-size on/off 124
-sizemax num 124
-sizemin num 125
-sizeflags attribute 125
Segment Control 125
-glue mainseg/curseg. 125
-IDEseg[ments] 126
-ma name=alias 126
-ra [*segment*]=*attribute*[,...] 127
-rn 129
-s name 129
-sb name [=*attribute* [,...]] 129
-sf *file*name.seg 130
-sg *newSegment* = [*oldSegment* [,...]] 130
-sn oldseg=newSeg 130
-single 130

-tmp directory . . . 130
Code Resource Control . . . 131
-coderesource . . . 131
-custom . . . 131
-da . . . 131
-drvr . . . 132
-fat filename . . . 132
-merge on/off . . . 132
-native . . . 133
-rsrcflags list . . . 133
-rsrcmerge . . . 133
-rsrcfar . . . 134
-rsrcmulti . . . 134
-rsrcname name . . . 134
-rsrcsegtype type . . . 134
-rsrcvirfuncs . . . 135
-rt *type*='*id*' . . . 135
-safe filename . . . 135
-xm c[oderesource] . . . 136
Code Fragment Control . . . 136
-ad num . . . 136
-appSub[FolderID] id . . . 136
-b . . . 137
-carr OSType . . . 137
-cfrg on/off . . . 137
-codeorder none/breadth[first]/depth[first]/segment/file=filename 137
-cv num . . . 138
-dv num . . . 138
-e name . . . 139
-export all/none . . . 139
-export [no]pragma . . . 139
-export name [, name, ...] . . . 139
-@export filename.exp . . . 140
-f filename . . . 140

-ib . . . 141
-init name . . . 141
-initbefore frag[,frag…] . . . 141
-librename importname=libname . . . 142
-m[ain] name . . . 142
-mergefragment filename . . . 143
-model CFMflat[rf]/CFMflatdf . . . 143
-name PEFname . . . 144
-sharedata . . . 144
-sortcode . . . 144
-stacksize num . . . 145
-term name . . . 145
-uv num . . . 145
-weakimport . . . 146
-weaklib frag[,frag…] . . . 146
-wi . . . 147
Output File Control . . . 147
-application . . . 147
-b . . . 147
-cv number . . . 147
-c creator . . . 147
-fc filecreator . . . 148
-ft filetype . . . 148
-G . . . 148
-library . . . 148
-libraryA4 . . . 148
-lLibname . . . 148
-Llibrary . . . 149
-mpwtool . . . 149
-o filename . . . 149
-sharedlibrary . . . 149
-t type . . . 149
-xma . . . 150
-xm a[pplication] . . . 150
-xm d[ropinaddition] . . . 150

-xm e[xecutable] . . . 150
-xml . . . 151
-xm l[ibrary] . . . 151
-xm libraryA4 . . . 152
-xm m[pwtool] . . . 152
-xms . . . 152
-xm s[haredLibrary] . . . 152
Linker Operation Control . . . 153
-fastlink on/off . . . 153
-help . . . 153
-link normal/fast[er]/slow[er] . . . 153
-maxerrors numbe . . . 154
-[no]mf . . . 154
-multidefs . . . 154
-nodefaults . . . 154
-noprejudge . . . 155
Diagnostic Message Control . . . 155
-cmap type . . . 155
-cexp type . . . 155
-d . . . 155
-map filename . . . 155
-mapIDE filename . . . 156
-msg *option* [,...] . . . 156
-nodup . . . 157
-p . . . 157
-version . . . 157
-w[arn] . . . 157
Debugging Control . . . 157
-csym '*creator*' . . . 157
-g . . . 158
-osym filename . . . 158
-sym on/off/full/fullpath/tdb/3.2/3.3 . . . 158
-tdb . . . 159

8 PowerPC Assembler 161
Overview of the PowerPC Assembler 161
Assembler options 161
-auxheader modtype 161
-blocksize size 162
-check 162
-d[efine] name[=value] 162
-depth value 162
-dialect type 162
-i pathname 163
-l 163
-lo listingName 163
-o objectName 163
-p 163
-t 163
-w 164

9 Metrowerks Disassemblers 165
Overview of Metrowerks Disassemblers 165
Disassembler options 165
-a4 165
-d 166
-fmt fmtparams 166
-format [no]x 166
-h 166
-help 166
-m name 167
-mix 167
-n 167
-noheader 167
-p 167
-sym on/off/full 167
-version 168
-xtables on/off 168

10 MPW Tools Compared 169

Overview of MPW Tools Compared . . . 169
C and MWC68K compilers . . . 170
CPlus and MWC68K compilers . . . 175
SC/Cpp and MWC68K compilers . . . 181
PPCC and MWCPPC compilers . . . 186
Optimization Differences . . . 187
MrC/MrCpp and MWCPPC compilers . . . 192
Optimization Differences . . . 192
Link and MWLink68K linkers . . . 199
PPCLink and MWLinkPPC . . . 203

11 CodeWarrior IDE and Command-Line Tools Compared 209

Comparison Overview . . . 209
Target Settings . . . 210
68K Target . . . 210
PPC Target . . . 225
Access Paths . . . 235
Language Settings . . . 238
C/C++ Language Settings . . . 238
C/C++ Warnings . . . 243
Pascal Language Settings . . . 246
Pascal Warnings . . . 250
PPCAsm . . . 252
Rez (and DeRez) . . . 254
Code Generation . . . 257
68K Processor Code Generation . . . 258
PPC Processor Code Generation . . . 261
68K Disassembler . . . 265
PPC Disassembler . . . 268
Linker . . . 270
68K Linker . . . 270
CFM68K Linker . . . 274
PPC Linker . . . 278
PPC PEF . . . 281

12 Pragmas and Command-Line Options 287
Overview of Pragmas and Options 287

Index 293

1 Introduction

This manual describes the CodeWarrior command-line tools—the compilers and linkers for the BeOS/BeBox platform and Macintosh Programmer's Workshop (MPW) environment from Apple Computer.

The Metrowerks command-line C/C++ and Pascal compilers, and the linker tools are the same compilers and linkers used in the Metrowerks CodeWarrior integrated development environment.

Read the Release Notes

Before using the command-line tools, read the release notes. They contain important information about new features, bug fixes, and any late-breaking changes.

The other topics in this introduction are:

- What's New in Command-Line Tools—a short review of new features
- Overview of Command-Line Tools—the chapters in this manual
- Starting Points—where to look for information

What's New in Command-Line Tools

For CodeWarrior Pro 3, the CodeWarrior IDE is significantly updated.

There are several new options for the MWC68K and MWCPPC compilers:

- `-make`—create a dependency rule suitable for a makefile, disables object file generation

- `-wchar_t on|off`—enable wchar_t as a built-in C++ type, corresponds to a new 'C++ Language' panel setting
- `-ec++ on|off`—enable Embedded C++; disables some C++ features such as templates and exceptions; corresponds to a new 'C++ Language' panel setting
- `-objc on|off`—enable Objective-C language extensions; corresponds to a new 'C++ Language' panel setting
- `-w [no]implicitconv` —warn when different types are implicitly converted, corresponds to a new 'C++ Warnings' panel setting
- `-w [no]notinlined`—warn when the compiler doesn't inline inline functions; corresponds to a new 'C++ Warnings' panel setting

The only new option for MWLinkPPC is:

- `-collapsereloads`—collapse unused TOC reload slots; removes unnecessary NOP instructions after function calls; works when linking applications and other kinds of code fragments; corresponds to a new 'PPC PEF' panel setting in the IDE.

Overview of Command-Line Tools

The chapters in this manual are:

- "About Command-Line Compilers and Linkers"—how to install MPW and the CodeWarrior command-line tools from the CodeWarrior CD.
- "MPW Basics"—a brief introduction to MPW, and how to use the CodeWarrior command-line tools within MPW.
- "Overview of Segmenting 68K Code"—an in-depth analysis of segmentation issues.
- "Overview of Metrowerks C/C++"—an option-by-option reference to the C and C++ compiler tools.
- "Overview of Metrowerks Pascal"—an option-by-option reference to the Pascal compiler tools.
- "Overview of Metrowerks Linkers"—an option-by-option reference to the linker tools.

- "Overview of Metrowerks Disassemblers"—an option-by-option reference to the disassembler tools.
- "Overview of MPW Tools Compared"—an option-by-option comparison of the CodeWarrior MPW Tools and other compilers and linkers.
- "Comparison Overview"—a cross reference guide to the IDE and command-line tool options.
- "Overview of the PowerPC Assembler"—a guide to the PowerPC assembler for the MPW environment.
- "Overview of Pragmas and Options"—a cross reference guide to the command-line tools options and the different #pragma compiler directives for the compilers and linkers.

Starting Points

If you need to install the CodeWarrior MPW Tools, see "Installation for MPW."

If you are unfamiliar with the MPW environment, you should read "MPW Basics." However, "MPW Basics" is not a user's guide to MPW. If you need to learn how to use MPW, refer to the various MPW manuals from Apple Computer. They are available in electronic form on the CodeWarrior CD, in the Documentation folder.

If you are working with 68K code and are concerned about segmentation, see "Overview of Segmenting 68K Code."

If you are an expert MPW user, you can read about the various compiler options in "Overview of Metrowerks C/C++." Linker options are found in "Overview of Metrowerks Linkers." Disassembler options are found in "Overview of Metrowerks Disassemblers." PowerPC assembler options are found in "Overview of the PowerPC Assembler."

If you are already familiar with one or another of several MPW tools—C, MrC/MrCpp, CPlus, SC/SCpp, PPCC, Link, and PPCLink—you will find an in-depth comparison of these tools to the CodeWarrior MPW Tools in "Overview of MPW Tools Compared."

If you are going to convert a CodeWarrior IDE project to a command-line environment, you'll want to read "Comparison Overview."

Finally, if you are interested in the various #pragma directives and their command-line equivalents, read "Overview of Pragmas and Options."

2

Getting Started

This chapter tells you how to install a CodeWarrior-savvy version of Macintosh Programmer's Workshop (MPW). CodeWarrior comes with a fully-functional version of MPW.

If you already have MPW, this chapter tells you how to add the CodeWarrior MPW tools to your environment. This process applies to the Mac OS.

If you own the BeOS tools, their installation will be covered in the Metrowerks Be product documentation.

About Command-Line Compilers and Linkers

The CodeWarrior development environment supports several platforms as targets for your code. Among them are the Mac OS and BeOS. Metrowerks provides tools you can use in the MPW environment to develop software for the Mac OS (or BeOS as a cross-compiler). The native BeOS command-line software development tools are available separately from the CodeWarrior product distribution.

The MPW Compiler and Linker tools are "fat" tools. This means that they run native on both 68K and PowerPC-based Macintosh computers, if you have a "fat" MPW Shell. Table 1.1 lists the Metrowerks MPW tools. There are compilers and linkers for PowerPC code and 68K code (for MacOS). There are disassemblers for PowerPC and 68K code. If you have the BeOS product, there is a compiler, disassembler and linker for PowerPC code development.

Table 2.1 CodeWarrior tools for MPW

Tool	Purpose
`MWPascal68K`	Compiles Pascal source code into 68K object code
`MWPascalPPC`	Compiles Pascal source code into PowerPC object code
`MWCPPC`	Compiles C/C++ source code into PowerPC object code.
`MWC68K`	Compiles C/C++ source code into 68K object code.
`MWLinkPPC`	Links files, including object code, into an executable PowerPC binary file.
`MWLink68K`	Links files, including object code, into an executable 68K binary file.
`MWDumpPPC`	Disassemble PowerPC code.
`MWDump68K`	Disassemble 68K code.
`mwcc`	BeOS command-line PowerPC C/C++ compiler.
`mwld`	BeOS command-line PowerPC linker.

Installation for MPW

To install the Metrowerks MPW tools, you need the latest Metrowerks CodeWarrior CD-ROM. It contains MPW, the tools for Mac OS and all the necessary headers and library files needed to use the MPW compiler and linker tools.

Installing MPW and the CodeWarrior Tools for MPW

CodeWarrior comes with a full version of the MPW environment that has been configured to include the CodeWarrior tools for

MPW, libraries, and examples. To use this preconfigured version of MPW, you simply install it on your computer.

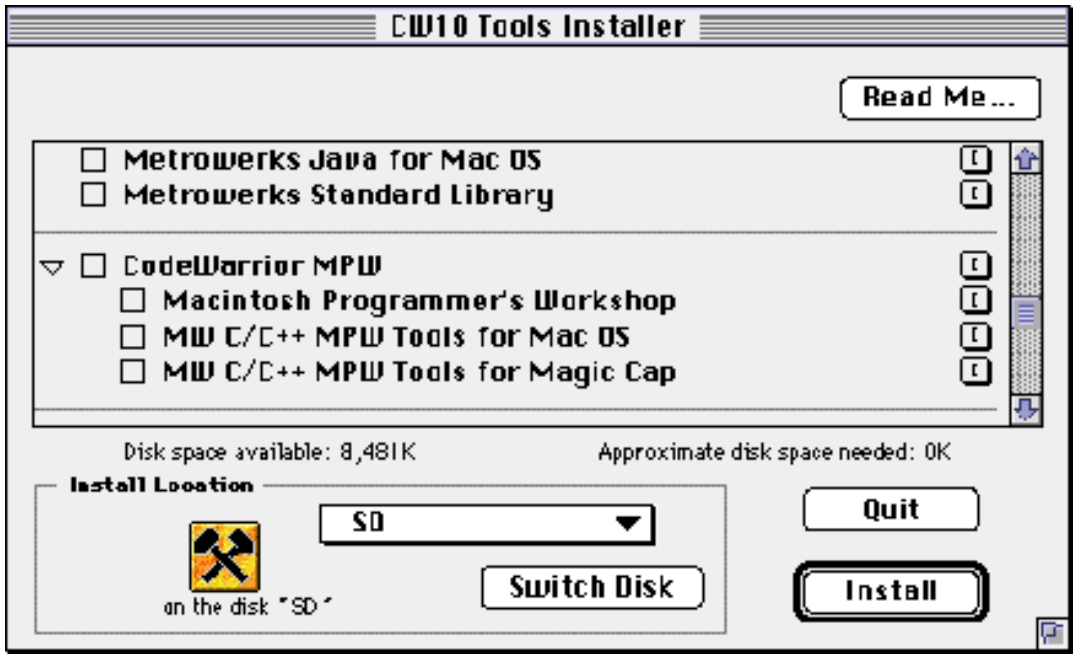
The CodeWarrior installer has options that allow you to install:

- Apple's MPW.
- Metrowerks MPW Tools for the Mac OS.

Run the installer and choose the items you wish to install.

Figure 2.1 shows the Installer options as they relate to MPW. (The actual installer is likely to vary in some details from this figure.) If you click the information button to the right of any option you can read information about the software installed with that option.

Figure 2.1 Installation options for MPW tools



If you choose the Macintosh Programmer's Workshop option, the Installer installs MPW only, not the Metrowerks tools for Mac OS development.

If you choose the Metrowerks MPW Tools for Mac OS option, the Installer installs the Metrowerks C/C++ compilers and linkers and other associated files. This option does not install MPW.

3

Programming with MPW

This chapter provides a brief overview of MPW. This section covers the most basic features required to write programs. MPW is a sophisticated development environment, and a complete description is beyond the scope of this manual. If you need to learn how to use MPW, refer to the various MPW manuals from Apple Computer. They are available in electronic form on the CodeWarrior CD. The manuals included are:

- Introduction to MPW.
- Building and Managing Programs in MPW.
- MPW Command Reference.
- Assembler for Macintosh with PowerPC
- MPW Asm Reference.

MPW Basics

This section discusses fundamental features of the MPW system.

The MPW Shell

The MPW Shell provides environment variables that allow the user to control the configuration of the MPW environment. Environment variables are an extension of the MPW Shell variables, and are exported by the Shell so that other programs have access to them.

A number of environment variables are set up in the startup files (`Startup`, `UserStartup`, and `Userstartup·Metrowerks`). Of these, the following are used by the MPW tools:

Name	Purpose
MWPInterfaces	Represents the folder containing Metrowerks Pascal interfaces files. Both compilers for PowerPC and 68K use this variable. To learn more about this variable and it's effects, refer to "Include file searching."
MWCIncludes	Represents the folder containing Metrowerks C system include files. Both CodeWarrior compilers for PowerPC and 68K use this variable. See also "Include file searching."
MWPPCLibraries	Represents the folder containing Metrowerks MPW PowerPC libraries. The Metrowerks PowerPC linker for MPW uses this variable.
MW68KLibraries	Represents the folder containing Metrowerks MPW 68K libraries. The Metrowerks 68K linker for MPW uses this variable.

Setting environment variables

A variable may be set using the MPW `set` command:

```
set MWCIncludes"{MPW}"Interfaces:MWCIncludes:
```

After the environment variable is set, it must be exported using the MPW `export` command:

```
export MWCIncludes
```

The value of a variable is used by enclosing the variable in curly braces ("{}"). For example to print the value of the variable `MWCIncludes`, the `Echo` command can be used:

```
Echo "{MWCIncludes}"
```

The *Introduction to MPW* Manual contains a complete list and description of environment variables.

Executing commands in MPW

The MPW Shell allows commands to be executed from the active window. Commands are executed by typing the command and pressing Enter or Command-Return. If a command is more than one line long, the whole command may be typed in, selected, and then executed by pressing the Enter key. After executing a command, the output produced by the command is placed in the active window.

Using CodeWarrior MPW tools

This section discusses how to use the CodeWarrior MPW tools from the MPW shell.

Preparing to use the CodeWarrior tools for MPW

When installing the CodeWarrior tools for MPW, many folders are copied to various places on your hard drive. If you change the location of these files, you will need to change the values of some of the environment variables. These variables are defined in `Startup`, `UserStartup` and `UserStartup·Metrowerks`.

Getting Help

To view a list of options for a given command, execute the help command followed by the command or tool for which you need assistance. For example, the following command lists the compiler options for the Metrowerks MPW 68K compiler tool for Mac OS:

```
help -f Metrowerks.Help MWC68K
```

Compiler options are detailed in "Overview of Metrowerks C/C++."

To view the list of options for the Metrowerks MPW 68K linker tool, execute the following command:

```
help -f Metrowerks.Help MWLink68K
```

Linker options are detailed in "Overview of Metrowerks Linkers."

Using the Tools

You use the CodeWarrior tools just like you would any other MPW tool.

To invoke a compiler tool, you execute a command from within an MPW Shell window (normally your Worksheet). The syntax is as follows:

```
MWC68K [option…] sourcefile
```

You start with the name of the tool. The `[option…]` part of the syntax is one or more compiler options. The *sourcefile* is the name of the source code file to be compiled. Note that when you use the compiler tool you must always include a source code file to be compiled.

Compiler options are detailed in "Overview of Metrowerks C/C++."

After a source file has been successfully compiled, invoke the linker tool from within an MPW Shell window. The syntax is as follows:

```
MWLink68K [option…] objectFiles
```

Once again you start with the name of the tool. The [*option…*] syntax is one or more linker options. The *objectFiles* syntax contains the names of one ore more object code files to be linked. Note that when you use the linker tool you must always include the name of an object code file to be linked.

Linker options are detailed in "Overview of Metrowerks Linkers."

Include file searching

By default, the compiler searches for system include files (files included as `#include <`*Header*`>`) in the directories specified by the `{MWCIncludes}` environment variable only. The compiler searches for normal include files (`#include "`*Header*`"`) in the current current working directory, and if it is not found, in the directories specified by `{MWCIncludes}`. To see what the current working directory is when in MPW, use the `directory` command.

See "Comparison Overview" for a discussion about the various options that can alter include file searching behavior. The options are -i, -i- and -cwd. For more information, refer to "Access Paths."

Example programs

The Metrowerks CodeWarrior CD contains example source code and MPW makefiles that demonstrate how to use the CodeWarrior MPW tools. There are examples for both 68K and PowerPC projects involving resources, applications, and PowerPlant.

Speeding Up MPW Builds

With CodeWarrior 9 and later, you can significantly reduce the time required to build a project using MPW.

There are no switches to set or special things to do to activate this feature. The trick is convincing Apple's Make tool to produce one build command that has all of your source files. The examples show how to do it.

The Make tool takes the following steps in the order given.

1. **Determines the target (your app or other output type).**
2. **Determines what the target depends on (the right side of the *ff*).**
3. **Determines if the target's dependencies are up to date.**

   Typically these are .o files or libraries.

4. **If any of the target's dependencies are out of date, it looks for "rules" to build them.**

   This is the heart of the trick for speeding up MPW. An object file's "rule" is of the type `c.o ƒ .c`. Make applies whatever follows this "rule" to any of the target's dependencies that need building. Usually the source file "rule" has a compiler command. Because any kind of MPW command is allowed, simply use the Set command to create a list of every file that needs compiling. Even though the "rule" itself doesn't build anything, Make is satisfied that it has done its job.

5. **Make issues the commands for the target.**

   After Make thinks it has built all of the dependencies or that they are otherwise up to date, Make issues the commands for the target. Here, order is important because Make will issue all of the target's *ƒƒ* "rules" in the order that they are in the make file. It is therefore important that you have the *ƒƒ* "rule" with the MWC68K or MWCPPC command before the *ƒƒ* "rule" with the MWLink68K or MWLinkPPC command.

   Listing 3.1 is a simple makefile that demonstrates this technique.

Listing 3.1 Speedy build with MPW

```
# Type "make all" into your worksheet and execute

Objects = main.c.o foo.c.o bar.c.o
COptions =

all ƒ start theApp

start ƒ $OutOfDate
set currentFiles ""

.c.o ƒ .c
set currentFiles "{currentFiles} ∂
{depDir}{default}.c"

theApp ƒƒ {Objects}
if "{currentFiles}" != ""
```

```
   MWC68K {COptions} {currentFiles}
end

theApp ƒƒ {Objects}
MWLink68K {Objects}
```

If you need to have more than one `COptions` variable for your files, Listing 3.2 shows you how.

Listing 3.2 Speeding up MPW with multiple COptions

```
# Type make all into your worksheet and execute.

Objects1 = main.c.o foo.c.o bar.c.o
Objects2 = foo_foo.c.o bar_bar.c.o
COptions1 = -sym off
COptions2 = -sym on

all ƒ start theApp

start ƒ $OutOfDate
set currentFiles ""

.c.o ƒ .c
set currentFiles "{currentFiles} ∂
{depDir}{default}.c"

theApp ƒƒ {Objects1}
if "{currentFiles}" != ""
   MWC68K {COptions1} {currentFiles}
   set currentFiles ""
end

theApp ƒƒ {Objects2}
if "{currentFiles}" != ""
   MWC68K {COptions2} {currentFiles}
   set currentFiles ""
end
```

```
theApp ff {Objects1} {Objects2}
MWLink68K {Objects1} {Objects2}
```

Look in the MPW materials on the CodeWarrior CD for more examples. In the MacApp tools folder you'll find some complex examples involving MacApp.

4

Segmenting 68K Code

This chapter provides an in-depth look at segmenting code for 68K applications and resources. Because the linker builds the final code, this chapter deals with the MWLink68K tool extensively.

Overview of Segmenting 68K Code

MWLink68K provides more options for controlling segmentation than MPW Link. Because each project type handles these options somewhat differently, each project type is discussed separately.

This chapter includes the following sections:

- Example Code—the code used as the basis for all the examples in this chapter.
- Segmenting with MPW and MWLink68K—basic segmentation issues.
- Advanced CodeWarrior Segmentation—advanced segmentation with MWLink68K.
- Segmenting Special Cases—segmentation considerations for applications, libraries, MPW tools, code resources, and drivers.
- Changing Resource Attributes—setting the attribute flags for code-related resources.

Example Code

All the examples included in this chapter are based on the same four source files: Main.c, Foo.c, Bar.c and FooBar.c. The code is shown in Listing 4.1.

Listing 4.1 Example Code

```
Main.c
/* prototypes */
void foo();
void bar();
void foobar();
#pragma segment Main
void main()
{
  foo();
  bar();
  foobar();
}
```

```
/*
 * Foo.c
 */

#pragma segment Foo
void foo()
{

}
```

```
/*
 * Bar.c
 */

#pragma segment Bar
void bar()
{

}
```

```
/*
 * FooBar.c
 */

#pragma segment FooBar
void foobar()
```

```
{

}
```

The basic make file to put this code together is shown in Listing 4.2.

Listing 4.2 Example make file

```
OutFile= Out
C    = MWC68K
Link= MWLink68K
COptions=
LinkOptions=
Objects = Main.c.o Foo.c.o Bar.c.o FooBar.c.o
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} ∂
    {Objects} ∂
    -o {OutFile}
```

Notice that there are no special options for either the compiler or the linker. The variables `COptions` and `LinkOptions` are empty. Each source file has its own `#pragma segment` directive.

Segmenting with MPW and MWLink68K

With MPW Link, the easiest way to control segmentation is to do nothing at all. If you had no `#pragma segment` directives, all of your object code would go into one segment. If the above example code did not have `#pragma segment` directives, you would generate the following CODE resources:

- `CODE 0 ""`
- `CODE 1 Main`

CODE 0 does not contain your object code, so it is outside the scope of this chapter. The CODE 1 segment would have all of your code, library code and—in the case of CW—runtime support for loading segments that is supplied by the linker.

MWLink68K and MWC68K support the standard MPW `#pragma segment` directive. Therefore, using the example code, you can generate a series of segments, one for each source file.

MWLink68K requires fast linking to be on for `#pragma segment` to be enabled. Fast linking is on by default. If you turn off fast linking, you will get unexpected results.

For example, look at Listing 4.3.

Listing 4.3 Using fastlink off

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -fastlink off ∂
    {Objects} ∂
    -o {OutFile}
```

With fast linking off, the MPW commands in Listing 1.3 generate the following CODE resources.

- `CODE 0 ""`
- `CODE 1 Main`

This is not what you would expect from looking at the example code. In contrast, the commands in Listing 4.4 generate the correct segments as indicated by the `#pragma segment` directives in the example code.

Listing 4.4 Using fast linking

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -fastlink on ∂
    {Objects} ∂
    -o {OutFile}
```

Note that fast linking is turned on. Therefore the commands in Listing 1.4 generate the following CODE resources:

- `CODE 0 ""`
- `CODE 1 Main`

- `CODE 2 Foo`
- `CODE 3 Bar`
- `CODE 4 FooBar`

That's what you would expect from the example code.

NOTE: MWLink68K has `Main` as the default name for the first code resource. This is the same name as MPW's Link. The IDE (CodeWarrior Integrated Development Environment) has `__%Main`.

Renaming Segments

With large and particularly mature projects, there are often many segment names that have outlived their usefulness. Instead of changing them in the source, you can remap them. MWLink68K supports `-sn` and `-sg` to remap segment names. The syntax for `-sn` is:

```
-sn oldname=newname
```

Any segment with `oldname` will be changed to `newname`.

NOTE: The IDE supports something like the `-sn` parameter in an auxiliary .SEG file. Please refer to the Targeting Mac OS manual. In a .SEG file for the IDE, the syntax requires quotes around the names, as in oldname=newname. Quotes are not required with MWLink68K unless the segment name has more than one word.

Based on the example source code listed at the start of this chapter, the MPW commands in Listing 4.5 would rename the Foo segment without changing the source code.

Listing 4.5 Renaming a segment

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -fastlink on ∂
    {Objects} ∂
    -sn Foo=Potato ∂
    -o {OutFile}
```

Note the use of the `-sn` option. As a result, the commands generate the following CODE segments:

- `CODE 0 ""`
- `CODE 1 Main`
- `CODE 2 Potato`
- `CODE 3 Bar`
- `CODE 4 FooBar`

If the new name matches an existing segment, the object code generated by both files is combined in a single segment. The commands in Listing 4.6 rename the Foo segment to Bar, which already exists.

Listing 4.6 Combining segments

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -fastlink on ∂
    {Objects} ∂
    -sn Foo=Bar ∂
    -o {OutFile}
```

Because the `-sn` option renames Foo to match Bar, the commands generate the following code segments:

- `CODE 0 ""`
- `CODE 1 Main`
- `CODE 2 Bar`
- `CODE 3 FooBar`

The segment Bar contains the object code for functions `foo()` and `bar()`.

Renaming Several Segments

The `-sg` linker option allows you to change several segment names to a single new name. The syntax is:

```
-sg newname[=oldname1[,oldname2]…]
```

NOTE: The `-sg` option puts the new name first, followed by the list of old names to be mapped to the new name. Compare that to `-sn`, which puts the old name first and the new name second.

The commands in Listing 4.7 rename the Bar and Foobar segments to Foo.

Listing 4.7 Renaming several segments

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -fastlink on ∂
    {Objects} ∂
    -sg Foo=Bar,Foobar ∂
    -o {OutFile}
```

Because Foo already exists, these commands generate the following code segments:

- `CODE 0 “”`
- `CODE 1 Main`
- `CODE 2 Foo`

In this case the Foo segment contains the object code for three functions, `foo()`, `bar()`, and `foobar()`.

You may have noticed from the syntax statement that the `-sg` option does not require an old name. The `-sg` parameter without an old name, as in `-sg` *newname*, renames all segments to *newname* and

concatenates all the object code. The commands in Listing 4.8 accomplish this trick.

Listing 4.8 Renaming all segments

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -fastlink on ∂
    {Objects} ∂
    -sg Foo ∂
    -o {OutFile}
```

Because there is no old name specified in the `-sg` option, these commands generate the following segments:

- `CODE 0 ""`
- `CODE 1 Foo`

The Foo segment contains all the object code from all source files.

Developers will often use pragma segments and remapping to organize the final build. Because they don't access the CODE segments by name, they prefer that the final product have nameless CODE segments. You can do this with `-rn`. Because the linker removes the name only after remapping, the output is the same. If `-rn` is used with a DA, the system accesses the first segment by name; `-rn` therefore removes all names except the first.

Until now, almost everything we have been talking about applies equally to both MPW Link and MWLink68K. In the next section we discuss options specific to the MWLink68K linker.

Advanced CodeWarrior Segmentation

MWLink68K provides three additional methods for controlling segments:

- `-single`
- `-sb`

- `-IDEseg`

NOTE: The `-sb` option is identical to `-s`.

These options can be used separately or with the standard methods already described in this chapter.

The syntax for the `-single` option is simple:

`-single`

This option puts all of the code into the first segment, even if fastlink is on and you have `#pragma segment` directives in your code. It is equivalent to `-sg Main`.

WARNING! If your project is large, you may need to set the far code and/or data options when compiling (`-model` far).

The syntax for `-sb` is also simple.

`-sb` *name*

This option puts all of the code that follows it into a segment with the specified name. `-sb` is the MWLink68K equivalent to the CodeWarrior IDE's notion of a segment in the project window. See the *IDE User Guide* for more information on segments in CodeWarrior projects.

The syntax for `-IDEseg` is also simple.

`-IDEseg`

The purpose of this option is to ensure that segment naming occurs in exactly the same way as generated by the CodeWarrior IDE. You have more options with the MPW tools than you have available in the IDE. The `-IDEseg` option allows different developers on the same project to use either MPW or CodeWarrior and generate com-

patible code. The CodeWarrior MWLink68K is essentially identical to Apple's Link with respect to segment remapping. With `-IDEseg`, segments created with `-sb` become a "hard" segment that neither `-sn` or `-sg` can rename. For the rest of this chapter, we will assume that you won't be using `-IDEseg`.

Renaming the First Segment

You can rename the segment in one of two ways. You can use options such as `-sg` or `-sn`, or you can use the CodeWarrior specific `-sb. If -sb` is positioned before any object files in the link map, then the first segment name will be changed to the name passed with `-sb`. Examine the commands in Listing 4.9.

Listing 4.9 Renaming Main

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -fastlink off ∂
    -sb Banana ∂
    {Objects} ∂
    -o {OutFile}
```

The `-sb` option tells the linker to create a single segment named Banana. You get the following CODE resources.

- `CODE 0 ""`
- `CODE 1 Banana`

Creating a Segment With No Name

The `-sb` option also lets you create a segment with no name at all if you want.

The commands in Listing 4.10 demonstrate how to create a segment with no name.

Listing 4.10 Creating a no-name segment

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -fastlink off ∂
    Main.c.o ∂
    -sb "" ∂
    Foo.c.o Bar.c.o FooBar.c.o ∂
    -o {OutFile}
```

These commands generate the following CODE resources:

- `CODE 0 “”`
- `CODE 1 Main`
- `CODE 2 “”`

In Listing 4.10 above, the commands list the separate object files to link. The Main.c.o file is linked first, and appears in the Main segment. Then the `-sb` command sets the next segment to no name. The remaining files are all linked into the second segment.

A Naming Frenzy

For one last extreme example of what you can do with MWLink68K, consider the commands in Listing 4.11.

Listing 4.11 The naming fool runs free

```
{OutFile} ƒƒ {Objects}
{Link} {LinkOptions} -fastlink on ∂
  -sb Banana Foo.c.o ∂
  -sb Bar FooBar.c.o ∂
  -sn Main=Banana ∂
  -sb FooBar Main.c.o ∂
  -sb Foo Bar.c.o ∂
  -o {OutFile}
```

These commands generate the following CODE resources:

- `CODE 0 “”`

- `CODE 1 Banana`
- `CODE 2 Bar`
- `CODE 3 FooBar`
- `CODE 4 Foo`

Because fastlink is on, each segment contains its self–named function (except Banana, which has the `main()` function). This is not a wise thing to do, but demonstrates the flexibility of `-sb`.

Segmenting Special Cases

In this section we discuss how segmentation affects various kinds of projects, including:

- Applications.
- Libraries.
- MPW Tools.
- Code Resources.
- Multi-Segment Drivers and Desk Accessories.

Applications

Applications support all of the directives discussed in this chapter. You may put an application in a single segment (keeping in mind that you may have to use the far model) or segment the application code as you wish.

Libraries

There is no segmentation in libraries, although libraries will store `#pragma segment` directives. They will not store segments created with `-sb`.

MPW Tools

Currently, all MPW tools are linked single-segment. You can change the first segment's name with `-sb` or `-sg newname`.

Refer to "Renaming the First Segment" and "Renaming Several Segments."

Code Resources

CodeWarrior supports single- and multi-segment code resources. You control the name of the main segment with the `-rsrcname` option. The default value for `-rsrcname` is "".

For more information, refer to "-rsrcname name."

If you are building a DA, provide the name without the leading '\0' character. CodeWarrior will insert the leading NULL for you.

If you are building a device driver, you are responsible for providing the leading period.

Listing 4.12 is an example of a basic make file for a code resource.

Listing 4.12 The basic code resource make file

```
OutFile= CROut
C      = MWC68K
Link   = MWLink68K

COptions= -a4 -model near
LinkOptions= -xm coderesource -rt FBar=128

Objects= Main.c.o Foo.c.o Bar.c.o FooBar.c.o
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} ∂
    {Objects} ∂
    -o {OutFile}
```

This make file tells the MWC68K compiler to use the a4 register for locating global data, and the near code model. It tells the MWLink68K linker to create a code resource, of type FBar and ID number 128.

Based on this make file, let's look at how to create single and multiple segment code resources.

Single-segment code resources

The commands in Listing 4.13 (based on the make file in Listing 4.12) show how to create a single-segment code resource.

Listing 4.13 Creating a single-segment code resource

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -rsrcname Coolness -fastlink off ∂
    {Objects} ∂
    -o {OutFile}
```

These commands generate the following resource:

```
· FBar 128 Coolness
```

The `-rsrcname` option sets the resource's name. The type and ID are set by commands shown in Listing 4.12.

The `-rsrcname` option is the only way to rename the first resource. The `-sb` option used for renaming segments does not work. Listing 4.14 shows how to do this.

Listing 4.14 Trying to name a code resource

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -fastlink off ∂
    -sb Coolness ∂
    {Objects} ∂
    -o {OutFile}
```

These commands generate the following resource:

```
· FBar 128 ""
```

Notice that the resource has no name, despite the attempt to use `-sb` to name the resource.

There is one other issue besides renaming that affects single-segment code resources. Even if you are linking into one segment, you must use the `-rsrcfar` option if you:

- have code larger than 32K, using `-model` far
- use an ANSI library (starting with CodeWarrior 6, the A4 versions of the ANSI libraries are far everything)
- have virtual functions (C++)

To learn more about this topic, refer to "-rsrcfar."

CodeWarrior supports linking all of the code in a code resource into a single segment with `-single`. There is no size restriction for a single segment if an appropriate code model is selected (`-model farCode` and/or `-model farData` for the compiler, and `-model far` for the linker).

As a result, there aren't many good reasons to link a code resource into several segments. However, if you must have segments, read on.

Multi-segment code resources

MWLink68K differs from the CodeWarrior IDE in that you can control the names and attributes of all segments, not just the first one. However, getting your `main()` function or entry point in the first segment—and naming that segment—can be a problem.

If you were writing a code resource that you load by name—say HardDiskEraser for example—Listing 4.15 shows the necessary commands.

Listing 4.15 Naming the first segment of a code resource

```
COptions= -a4 -model near
LinkOptions = -xm coderesource -rt FBar=128 -fastlink on
Objects= Main.c.o Foo.c.o Bar.c.o FooBar.c.o
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -rsrcmulti -rsrcname
                HardDiskEraser ∂
```

```
    {Objects} ∂
    -o {OutFile}
```

Because fastlink is on, and each source file has a `#pragma segment` directive, these commands generate the following resources of type FBar:

- `FBar 128 HardDiskEraser`
- `FBac 129 Foo`
- `FBac 130 Bar`
- `FBac 131 FooBar`

The HardDiskEraser segment has startup and dispatch code provided by the linker, *and* the object code for `main()`.

You might wonder how the code for `main()` ended up in a segment named HardDiskEraser. The `#pragma segment Main` directive in the source code puts the object code for that file into the first segment. The `-rsrcname` simply renames Main to HardDiskEraser afterwards.

NOTE: It is a good thing to remember that the linker renames the first segment to the `-rsrcname` after it has obeyed pragma and remapping directives.

If you use `-IDEseg`, the `-rsrcname` option is the only way to rename the first resource. Listing 4.16 demonstrates one technique that you might think would work, but it fails.

Listing 4.16 The wrong way to rename the first segment

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -rsrcmulti -rsrcname
          HardDiskEraser -IDEseg ∂
    {Objects} ∂
    -sn Main=HardDiskEraser ∂
    -o {OutFile}
```

These commands generate the following resources:

- `FBar 128 HardDiskEraser`
- `FBac 129 HardDiskEraser`
- `FBac 130 Foo`
- `FBac 131 Bar`
- `FBac 132 FooBar`

Oops. This is something that you definitely don't want. The object code for `main()` is put in a segment originally named Main but changed to HardDiskEraser by the `-sn` command. When the `-rsrcname` command executes—*after* it has obeyed pragma and remapping directives—there is no longer a Main segment to rename. So it creates the first segment and names it HardDiskEraser.

As a result, the FBar 128 resource contains startup code provided by the linker. The object code for `main()` is in the second HardDiskEraser segment, a result that is probably not what you wanted to happen.

Starting with CodeWarrior 7, MWLink68K behaves more like Apple's Link tool would with the commands in Listing 4.16. If you do not use `-IDEseg`, you get only one segment named HardDiskEraser.

The `-IDEseg` option is provided for the situation where a single project has some developers who use the MPW tools, and others who use the CodeWarrior IDE. In that case, to ensure that all developers produce the same segmentation, use `-IDEseg`. To solve the problem in Listing 4.16, if you need to use `-IDEseg`, remove the `-sn Main=HardDiskEraser` command. Then the listing produces only one segment named HardDiskEraser.

Multi-Segment Drivers and Desk Accessories

Multi-segment drivers and Desk Accessories are different from any other code resource type (including single segment drivers and Desk Accessories) because CodeWarrior creates a special segment for them. This segment takes the name and ID of your resource, but it has no code from your source. It is basically a dispatcher.

Trying to name the segments in multi-segment drivers and Desk Accessories can lead to the kinds of problems we saw in Listing 4.16. Listing 4.17 shows the necessary commands and what happens as a result.

Listing 4.17 Creating a multi-segment DA

```
COptions= -a4 -model near
LinkOptions -xm coderesource -rt DRVR=128 -fastlink on
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -rsrcmulti -rsrcname
              HardDiskEraser -da ∂
    {Objects} ∂
    -o {OutFile}
```

Note the `-da` option for the linker. These commands create the following resources:

- `DRVR 128 HardDiskEraser`
- `DRVc 129 Main`
- `DRVc 130 Foo`
- `DRVc 131 Bar`
- `DRVc 132 FooBar`

The first segment has linker-supplied code. The object code for `main()` is in segment 129. How did segment 129 get its name? It isn't because there was a `#pragma segment Main` in Main.c. The default first segment is always Main.

In order to change the name of the second segment—the one named Main in the previous example—you have to use a little sleight of hand.

Listing 4.18 Renaming the Main segment

```
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -rsrcmulti -rsrcname
              HardDiskEraser -da ∂
    -sb Potato ∂
```

```
-sn Main=Potato ∂
{Objects} ∂
-o {OutFile}
```

These commands do the right thing. They create the following resources:

- `DRVR 128 HardDiskEraser`
- `DRVc 129 Potato`
- `DRVc 130 Foo`
- `DRVc 131 Bar`
- `DRVc 132 FooBar`

This is what you want to accomplish. The code for the `main()` function is in the Potato segment, the first code-based segment. The `-sn` command remapped the Main segment code into the new Potato segment.

NOTE: You can't `#pragma segment` into one of the dispatcher segments because it is named after the `#pragma segment` directives are honored.

Custom headers

Using custom headers with the MWLink68K linker is a little more flexible than with the IDE. For one thing, you can use `-da` and `-custom` at the same time. Apart from that you must follow the restrictions laid out in the "Creating Code Resources" chapter in the *Targeting Mac OS* manual. Listing 4.19 shows the MPW commands that do the job.

Listing 4.19 Using custom headers

```
COptions= -a4 -model near
LinkOptions= -xm coderesource -rt FBar=128 -fastlink on
{OutFile} ƒƒ {Objects}
  {Link} {LinkOptions} -rsrcmulti -rsrcname
          HardDiskEraser -da -custom ∂
```

```
header.c.o ∂
CustomCRStartup.lib ∂
{Objects} ∂
-o {OutFile}
```

These commands create the following resources:

- `FBar 128 \0HardDiskEraser`
- `FBac 129 Foo`
- `FBac 130 Bar`
- `FBac 131 FooBar`

The HardDiskEraser segment contains `main()`, as well as header.c.o and CustomCRStartup.lib.

NOTE: Because this uses a custom header, no dispatcher segment is made for you.

Changing Resource Attributes

MWLink68K supports the standard MPW parameter `-ra` to set attributes. However, MWLink68K also provides a special parameter that sets the resource attributes for the first segment of a code resource, `-rsrcflags`.

By default, applications and MPW tools have their first segment set to protected, locked and preload. If your application has other segments, by default, they are set to protected and purgeable. Code resources default all segments to no attributes.

Using -ra

You can modify the attributes with the following syntax:

```
-ra [segname]=attr[,attr…]
```

The `attr` is an attribute from the following list:

- resSysHeap
- resPurgeable
- resLocked
- resProtected
- resPreload
- a hex or decimal attribute you determine

If `segname` is not supplied, the attributes apply to all segments except CODE 0 and CODE 1 for applications and MPW Tools, or for all segments except the first segment in a code resource.

If `segname` is supplied, the attributes apply to that segment and override other settings.

Here are some examples:

```
-ra Main=40
-ra Foo=resPurgeable
-ra =$28
```

and for code resources

```
-ra HardDiskEraser=resLocked -rsrcname HardDiskEraser
-ra =resPurgeable -rsrcname HardDiskEraser
```

Naming a specific segment with `-ra segname=attr` overrides other attribute settings. You can use `-ra Foo=$28` in the same link command as `-ra =resPurgeable`. This combination sets segment Foo to $28 and all other segments, except 0, 1 and Foo to `resPurgeable`.

Setting Attributes for Code Resources

Code resources support all of the above uses of `-ra` but also allow `-rsrcflags` to change the attributes of the first segment. The `-rsrcflags` option only changes the attributes of code resources.

The syntax is:

```
-rsrcflags list
```

The `list` parameter is a comma-separated list of attributes from the following list of possibilities:

- system
- purgeable
- locked
- protected
- preload

NOTE: `-ra` and `-rsrcflags` use different names for the specific attributes.

There is a special twist for multi-segment code resources, however. If you set the attributes of the first segment—with either `-ra` or `-rsrcflags`—the default attributes for other segments change as well.

If you modify the attributes of the first segment, the remaining segments' attributes are modified according to the following algorithm:

```
otherSegmentsAttributes =
      firstSegmentsAttributes &(system | protected)
```

Again, using `-ra =attr` overrides `otherSegmentsAttributes` as does naming a specific segment with `-ra segname=attr`.

5

Metrowerks C/C++

This chapter lists and describes the compiler options. Options control the way the compiler translates source code, generates object code, and communicates with the programmer.

Overview of Metrowerks C/C++

The Metrowerks C/C++ command line compilers —MWCPPC (Mac OS), mwcc (BeOS) and MWC68K (Mac OS)—translate C and C++ source code into PowerPC or 68K object code. Object code generated by these compilers is processed by the Metrowerks command line linkers—MWLinkPPC (Mac OS), mwld (BeOS) and MWLink68K (Mac OS)—to produce executable objects like applications, code resources, and shared libraries.

Each compiler has various options that you can set. If you are familiar with the CodeWarrior development environment you will see that many of these options correspond to settings in the CodeWarrior Preferences dialog. "Comparison Overview" shows the relationships.

This chapter organizes options into these groups:

- Code Generation Control—options that affect how object code is created and arranged in the compiler's output file.
- Preprocessing and Precompiling—options that affect preprocessor macro definition settings and precompiled file control.
- Language Control—options that affect how the compiler translates source code.
- Input File Control—options that affect file inclusion.
- Diagnostic Message Control—options that determine how the compiler communicates warnings, errors, and progress information to the user.

- Debugging Control—options that tell the compiler to generate object code for use with debuggers.
- Profiling Control—options that tell the compiler to generate object code for use with the CodeWarrior Profiler.

In all cases, text in braces—the [and] characters—is optional. The braces themselves never appear in the actual command. For example, you could write `-str[ings] pool` as either `-strings pool` or `-str pool`.

Where an option has several possible permutations, the possibilities are separated by a slash—the / character. For example, `-sym on/off/full/fullpath` reflects the fact that the `-sym` command may be followed by one or more of four possibilities. So you might have `-sym on`, `-sym off`, `-sym full`, or `-sym fullpath`.

In cases where you provide a variable parameter, like a file name, that item is in italic text. For example, `-precompile` *filename* means you must provide a file name. The explanation that accompanies each compiler option makes it clear what you must provide.

Code Generation Control

-a4

Mac OS

For MWC68K only.

Tells the compiler whether to use the 68K processor's A4 register for global data references.

Use this option when creating 68K code resources.

See also `-coderesource` on page 14.

-[no]a6

For MWC68K only.

Tells the compiler whether to generate function stack frames based on the A6 register. The default is `-a6`.

With A6 stack frame generation off, `-noa6`, the compiler does not generate stack frames when they are unnecessary. This results in slightly smaller, faster object code that is harder to trace in a low-level debugger.

This option is equivalent to `#pragma a6frames`.

Mac OS

Use the `-a6` option when generating object code that will be used with the CodeWarrior debugger.

-align array[members]

This option aligns array elements to make references to arrays faster. The array may occupy more memory. With array alignment off, arrays may take up less memory, but references to array elements may be slower.

Note that this option is only effective on arrays that are not already aligned in memory.

The default is no array alignment.

This option is equivalent to the `#pragma align_array_members` directive.

-align mac68k/mac68k4byte/power/packed

This option specifies alignment for data structures. With `mac68K`, the compiler aligns an item on a 2-byte boundary, unless it is only 1 byte long. The `-align mac68k4byte` option specifies 68000-family 4-byte alignment for data structures.

This option is equivalent to the `#pragma align` directive.

Mac OS

The `power` option specifies PowerPC alignment for data structures. This option aligns data on its natural boundary. In other words, a character is aligned on a 1-byte boundary, a 16-bit integer is aligned on a 2-byte boundary, and so on. This alignment applies recursively to structured data and arrays containing structured data. So, for example, an array of structured types containing an 8-byte floating point member must be aligned on an 8-byte boundary.

The default for MWCPPC is `-align power`. The default for MWC68K is `-align mac68k`.

The `packed` option can be specified for packed alignment of data structures. Packed structures can cause incompatibilities with existing libraries and address errors on certain processors, such as the Motorola 68000.

-b

For MWC68K only.

Tells the compiler whether to put string constants into instruction code instead of data. This option also tells the compiler to generate references to string constants and functions relative to the 680x0 processor's program counter (PC register) if they are in the same segment.

The default is to generate references relative to the A4 or A5 registers of the 680x0 processor (depending on the use of the `-a4` option).

See also `-a4`.

This option is equivalent to the `#pragma pcrelstrings` directive.

-b2

For MWC68K only.

Has the same functionality as the `-b` option. In addition it places identical string constants in the same memory space.

This option is equivalent to the `#pragma pcrelstrings` directive.

-Cpp_exceptions on/off

Mac OS and BeOS

Enables or disables true C++ exception handling.

This option

```
-Cpp_exceptions on
```

turns on true C++ exception handling. This is the default.

This option

```
-Cpp_exceptions off
```

is equivalent to the `-directdestruction` option.

See also `-directdestruction`.

This option is equivalent to the `#pragma exceptions` directive.

-d0

MWC68K only.

Tells the compiler to return pointer types in the D0 register of the 680x0 processor instead of the A0 register.

This option is equivalent to the `#pragma d0_pointers` directive.

Mac OS

Use this option when generating object code that deals with the MPW Shell or MPW libraries.

-directdestruction

Mac OS

This is equivalent to -Cpp_exceptions off. You will get a warning if you pass `-directdestruction` since the option no longer does

anything. This avoids confusion because the #pragma direct_destruction is no longer supported.

For C++ source code. Tells the compiler to generate C++ object code that destroys local and temporary objects with a direct destructor call. Using this option prevents the use of the `__local_destructor_chain` mechanism to destroy these class objects.

The default is no direct destruction.

This option is equivalent to the `#pragma direct_destruction` directive.

NOTE: PowerPlant's exception handling (based on `__longjmp_ldestr`) will not work when this option is used.

-dsize 64/80

For MWC68K only.

Specify the size of the double data type. You specify the number of bits. To specify 8-byte `doubles`, use `-dsize 64`. To specify 10-byte `doubles`, use `-dsize 80`. The default is `-dsize 64`. 10-byte `double` size is the same size used by the SANE `extended` data type.

This option is equivalent to the `#pragma IEEEdoubles` directive.

-ext extension

Specifies the filename extension of the compiler's output file.

If *extension* has a leading period (`.`) then *extension* is added to the end of the name of the compiler's output file, which contains object code. If *extension* has no leading period then *extension* replaces the output filename's extension.

For example,

```
MWC68K SockSorter.c -ext .mw.c.o
```

specifies an output file named SockSorter.c.mw.c.o.

```
MWC68K SockSorter.c -ext lib
```

specifies an output file named SockSorter.lib.

By default, *Extension* is "`.o`". The `-ext` option is ignored if the `-o name` option is used.

-enum int/min

Tells the compiler to allocate either `int`-sized or the smallest possible size memory cells for enumeration types.

This option is equivalent to the `#pragma enumsalwaysints` directive.

Mac OS

The default for Mac OS is `-enum min`.

BeOS

The default for BeOS is `-enum int`.

-fatext

Mac OS

Changes the default output name extension. For MWCPPC, from .o to .ppc.o. For MWC68K, from .o to .68k.o. If an actual output file is passed with `-o name`, `-fatext` does nothing.

-fdt num

Mac OS

This option sets the data threshold above which near data gets put into far data space. The default for *num* is 256 (bytes). You should try a smaller numer like 32 if you run out of near data space. This option is only relevant with `-model neardata`.

See also `-model neardata`.

-force_active

Mac OS

For MWC68K only.

Tells the compiler to mark all code so that it will not be removed by the linker, even if the code is "dead."

See also `the discussion in` "-dead on/off/code/initcode."

This option is equivalent to the `#pragma force_active` directive.

-fp_contract on/off

Mac OS and BeOS

For MWCPPC and mwcc only.

Tells the compiler to enable or disable PowerPC floating-point contraction instructions (for example, `FMADD`, `FMSUB`). The default is `-fp_contract on`.

This option is equivalent to the `#pragma fp_contract` directive. To learn more about pragma statements, refer to the *C Compiler Guide*.

-fp_library on/off

For MWC68K only.

This option allows support for a new non-SANE/non-68881 floating-point run time model. This model emulates IEEE floating point calculations in software. A 32-bit floating point quanity is a `float`, and a 64-bit floating quantity can be a `short double`, `double`, or `long double`.

In order to use this options, you must link with the run time library named `fp68k.lib`.

-fp_pilot_traps on/off

PalmOS

For MWC68K only.

This option will use the PalmOS system traps instead of the fp68K.lib floating point library.

-inline on/all/off/none/static/auto

Mac OS and BeOS

Tells the compiler whether to generate C++ functions defined with the `inline` directive directly in the code, and not as real functions. This saves function calling overhead and can result in slightly faster code.

The default is `-inline on`. The "on" and "all" options are synonyms. The "off" and "none" options are also synonyms.

The "static" option tells the compiler to generate functions defined with the `inline` directive as static functions.

This option can be activated or deactivated with the `#pragma dont_inline, #pragma static_inline or #pragma auto_inline` directives.

-intsize 2/4

MWC68K only.

Specifies the size of the `int` data type, in bytes. To specify 2-byte `int`s, use `-intsize 2`. To specify 4-byte `int`s, use `-intsize 4`. The default is `-intsize 4`.

This option is equivalent to the `#pragma fourbyteints` directive.

Mac OS

When creating an MPW tool, or using source code that is compiled for both 68K and Power Macintosh computers, use `-intsize 4`.

-indirect

Mac OS

MWC68K only.

This option forces the access of all global references by indirection. This option is off by default.

-ldsize 64

Mac OS

MWCPPC only.

Tells the compiler to use 8-byte doubles (64 bits). For PowerPC, only 8-byte doubles are currently supported.

-make

Mac OS

MWC68K and MWCPPC only.

Tells the compiler to create a dependency rule suitable for a make-file, and disables object file generation.

-mc68020

Mac OS

MWC68K only.

Tells the compiler to generate instructions specifically for the Motorola 68020, 68030, and 68040 processors.

Using this option creates object code that cannot be run on 68000 processors. By default, MWC68K generates object code compatible with all Motorola 68000-family processors.

This option is equivalent to the `#pragma code68020` directive.

-mc68881

MWC68K only.

Tells the compiler to generate code for mathematical operations for a Motorola 68881 or Motorola 68882 math coprocessor instead of calling library routines.

By default, MWC68K compiles mathematical operations into library calls.

This option is equivalent to the `#pragma code68881` directive.

Mac OS

Using this option creates object code that cannot be run on 68K Macintosh computers that do not have math coprocessors.

-mpwc

MWC68K only.

Generate code using MPW calling conventions. This option is equivalent to `#pragma mpwc` C/C++ compiler directive. Refer to the pragma section of the *C Compiler Guide* for more information.

-model CFMflat[rf]/CFMflatdf

Mac OS

MWC68K only.

Tells the compiler to generate code to create a CFM68K object file. This option can be used in conjunction with other `-model` options. For the MWC68K tool all these options so the same thing.

-model farCode/nearCode/codeSmart

MWC68K only.

The “farCode” option tells the compiler to always generate 32-bit references for executable code, even if a 16-bit reference is possible.

The “nearCode” option tells the compiler to always generate 16-bit references for executable code. If a 32-bit reference is necessary, the linker will post an error.

The `-model codeSmart` option generates 16-bit references whenever possible. If an object is beyond a 16-bit range, the compiler generates a 32-bit reference.

The default is `-model nearCode`.

These options may also be manipulated with the `#pragma far_code, #pragma near_code and #pragma smart_code` directives.

-model farData/nearData

MWC68K only.

The `-model farData` option removes the 64K limit on global data and stores global data in far data space instead of near data space. When used, there is no limit on global data. The generated code will also be slightly bigger and slower.

The `-model nearData` option tells the compiler to generate 16-bit references to data. This option limits global data to 64K

The default is `-model nearData`.

This option may also be manipulated with the `#pragma far_data` directive.

-model farStrings/nearStrings

MWC68K only.

The `-model farStrings` option tells the compiler to generate 32-bit references to string literals.

The `-model nearStrings` option tells the compiler to generate 16-bit references to string literals. This option limits literal character string storage to 64K.

The default is `-model nearStrings`.

This option may also be manipulated with the `#pragma far_strings` directive.

-model farVTables/nearVTables

Mac OS

MWC68K only.

The `-model farVTables` option tells the compiler to generate 32-bit references to C++ virtual function tables.

The `-model nearVTables` option tells the compiler to generate 16-bit references to C++ virtual function tables. This option limits virtual function tables to a maximum of 64K.

The default is `-model nearVTables`.

This option may also be manipulated with the `#pragma far_vtables` directive.

-model far/near

MWC68K only.

The `-model far` option tells the compiler to use `-model farCode`, `-model farData`, `-model farStrings`, and `-model farVTables`. This would be equivalent to the following pragmas:

```
#pragma far_data on
#pragma far_code on
#pragma far_strings on
#pragma far_vtables on
```

The `-model near` option tells the compiler to use `-model nearCode`, `-model nearData`, `-model nearStrings`, and `-model nearVTables`.

The default is `-model near`.

This option may also be manipulated somewhat with the `#pragma near_code` directive.

-nodirectdestruction

BeOS

mwcc only.

This is equivalent to -Cpp_exceptions on.

Disables direct destructor call to destroy local objects.

-o name

Tells the compiler the name of the file or directory in which to save object code in the CodeWarrior object file format for 68K or PowerPC. If this option is not specified, or if you provide a directory name, the default file name for the object code file is the source code filename with a ".o" extension.

See also `-c`, `-ext extension`.

BeOS

For the BeOS, it is possible to pass several files on the mwcc command line without explicitly invoking the mwld linker. An example of this would be:

mwcc filename1.cpp filename2.cpp -o MyBeApp

-opt option[, ...]

The `-opt` option controls compiler optimizations. The available optimizations differ for MWC68K and MWCPPC. In either case you

specify a list of optimization options from the possibilities listed below. Multiple options must be separated from each other by commas, with no white space. You must supply at least one option. The default for both compilers is `-opt on`. The effect of `-opt on` is slightly different, because the two compilers have different possible optimizations. The MWC68K compiler does not have scheduling and level-related options.

For MWC68K, the possible optimizer options are:

- `on`—turns on all optimizations. This is equal to `-opt global,peep, speed`.
- `off`—turns off all optimizations.
- `all`—turns on all optimizations. This is the same as `-opt on`.
- `full`—same as `all`.
- [no]`cse`—This option is equivalent to `-opt global on/off`
- `[no]color`—turns register coloring on or off (68K code only). The default is on.
- [no]`global`—tells the compiler to generate object code with or without global optimization and use register allocation.
- [no]`peep`—tells the compiler to generate object code with or without peephole optimization. Peephole optimization performs small, local optimizations to improve execution speed. Peephole optimization eliminates redundant compare instructions and improves branch sequences.
- `size/speed`—tells the compiler to generate object code that is optimized for a smaller size or greater speed. These cannot both be turned on, they are alternatives. The compiler optimizes for smaller object code size at the expense of slower execution. The compiler optimizes for faster execution speed at the expense of larger object code size. You always have either `size` or `speed` selected. The default is `speed`.

This option may also be manipulated with the `#pragma scheduling`, `#pragma optimization_level`, `#pragma optimize_for_size`, `#pragma peephole` and `#pragma global_optimizer` directives.

Mac OS and BeOS

For MWCPPC and mwcc, the possible optimizer options are:

- `on`—turns on all optimizations, level 1. This is equal to `-opt global, peep,schedule,l1,speed`.
- `off`—turns off all optimizations.
- `all`—turns on all optimizations, level 4. This is the same as `-opt global,peep,schedule,l4,speed`.
- `full`—same as `all`.
- [`no`]`global`—tells the compiler to generate object code with or without global optimization and use register allocation.
- [`no`]`peep`—tells the compiler to generate object code with or without peephole optimization. Peephole optimization performs small, local optimizations to improve execution speed. Peephole optimization eliminates redundant compare instructions and improves branch sequences.
- [`no`]`schedule`—tells the compiler to generate object code with or without instruction scheduling optimization. Instruction scheduling means that the compiler arranges certain sequences of machine language instructions to take advantage of the PowerPC's pipeline architecture. The `-schedule` option is the same as `-opt schedule601`.
- `schedule601`, `schedule603`, or `schedule604`—tunes scheduling for the PowerPC processor specified. The `schedule601` option is the best compromise for mixed PowerPC machines.
- `l1`, `l2`, `l3`, or `l4`— sets the global optimization level. `l1` offers the least optimization, while `l4` offers the most optimization.
- `size/speed`—tells the compiler to generate object code that is optimized for a smaller size or greater speed. These cannot both be turned on or off, they are alternatives. The compiler optimizes for smaller object code size at the expense of slower execution. The compiler optimizes for faster execution speed at the expense of larger object code size. You always have either `size` or `speed` selected. The default is `speed`.

-O

BeOS

There are several case-sensitive optimization options that can be passed to the mwcc compiler when developing for the BeOS. All of these options begin with the capital letter 'O' as shown in Table 5.1.

Table 5.1 C/C++ Optimization Options for mwcc (BeOS)

Command Line Option	Description
-O	enable PowerPC 603 instruction scheduling
-O0	suppress all optimizations
-O1	same as -opt peep
-O2	same as -opt schedule603
-O3	same as -opt global
-O4	same as -opt global,peep,schedule,l1,speed
-O5	same as -opt global,peep,schedule,l2,speed
-O6	same as -opt global,peep,schedule,l3,speed
-O7	same as -opt global,peep,schedule,l4,speed
-Op	same as -opt speed
-Os	same as -opt size

This option may also be manipulated with the `#pragma scheduling`, `#pragma optimization_level`, `#pragma optimize_for_size`, `#pragma peephole` and `#pragma global_optimizer` directives.

-rostr

Mac OS and BeOS

MWCPPC and mwcc only.

Tells the compiler to store string constants in the code section of the generated PEF file, instead of the data section. All objects in the code section are read-only. By default, the compiler stores string constants in the data section.

See also `-str[ings] readonly`.

This option may also be manipulated with the `#pragma readonly_strings` directive.

-RTTI on/off

Mac OS and BeOS

Enables run-time type information for C++ code. RTTI is enabled by default for BeOS and disabled by default for Mac OS.

This option may be manipulated with the `#pragma RTTI` directive.

-s name

MWC68K only.

Tells the compiler to insert the equivalent of a

```
#pragma segment name
```

at the beginning of the source code file to be compiled.

Specifying

```
-s ""
```

is equivalent to specifying

```
-s Main
```

-som

Mac OS only

The compilers can now emit SOM (System Object Model) files. This Direct-To-SOM compilation feature supports Apple Computer's SOM objects implementation for both 68K and PowerPC.

This option may be manipulated with the `#pragma direct_to_som` directive.

-som_env_check

Mac OS only

The compilers can now emit SOM (System Object Model) files. This Direct-To-SOM compilation feature supports Apple Computer's SOM objects implementation for both 68K and PowerPC. This option enables automatic SOM environment and new allocation checking.

This option turns on the `-som` option if it is not already on.

This option may be manipulated with the `#pragma SOMCheckEnvironment` directive.

-str[ings] dontreuse

Tells the compiler not to reuse identical character string literals.

This option may also be manipulated with the `#pragma dont_reuse_strings` directive.

-str[ings] pool

Tells the compiler to place character string literals into a single data object.

This option may also be manipulated with the `#pragma pool_strings` directive.

Mac OS and BeOS

MWCPPC and mwcc only.

When creating a PEF file (an application, shared library, or code resource), this option reduces the Table of Contents size but may increase code size.

-str ro

BeOS

mwcc only.

Make string constants read-only. The default is off.

This option may also be manipulated with the `#pragma readonly_strings` directive.

-str[ings] readonly

Mac OS

MWCPPC only.

Tells the compiler to store character string literals in the fragment's Code Section instead of the Data Section. All objects in the Code Section are read only.

This option is equivalent to `-rostr`.

This option may also be manipulated with the `#pragma readonly_strings` directive.

-toc_data on/off

Mac OS and BeOS

MWCPPC and mwcc only.

Tells the compiler whether to store small (that is <= 4 bytes) static data in the TOC. This makes all references to the data smaller and faster.

The default is `-toc_data on`.

This option may also be manipulated with the `#pragma toc_data` directive.

Preprocessing and Precompiling

-check precomp[flags]

Tells the compiler to make sure the settings in precompiled header files match the settings for the currently compiling source code.

For MWC68K, use this option when creating a precompiled header. When `used`, the compiler ensures that the precompiled header's settings for `double` size (8-byte or 12-byte), `int` size (2-byte or 4-byte) and Mac OS SANE or Motorola 68881 floating point math settings correspond to the project's preferences.

Precompiled headers that do not rely on these settings should not use this option. Precompiled headers that depend on these settings (like the Standard ANSI C and ANSI C++ library header files that use `int` and `double`) should use this option.

See also `-precompile filename`

-d name [= [value]]

Tells the compiler to define a preprocessor symbol with the name *name*. The default value for the defined symbol is 1. Use the optional item, *value*, to specify a value other than 1.

For example, entering

```
-d debug_on
```

is equivalent to placing

```
#define debug_on 1
```

in the first line of the source code file being compiled.

Entering
```
-d version=2.01
```

is equivalent to placing
```
#define version 2.01
```

at the beginning of source code file being compiled.

Entering
```
-d debug_on=
```

is equivalent to
```
#define debug_on
```

-D name [= [value]]

BeOS

This option is the same as -d *name[=[value]]*

-dump[c] filename

Same as the `-precompile filename` option.

This option may also be manipulated with the `#pragma precompile_target` directive.

-e

Sends preprocessor output to standard output without generating object code. Use this option to verify preprocessor directives and definition expansions.

See also `-c`.

-load filename

Same as `-prefix filename`. See also `-i directory`.

-[no]mf

Mac OS

Tells the compiler whether to use Finder memory (the Process Manager heap) when necessary. Use of the `-mf` switch will dramatically save memory in the MPW Shell's partition. The default is `-mf`.

The `-nomf` option tells the compiler to *never* use MultiFinder memory. The Shell's partition must hold all data.

-[not]once

The `-once` option tells the compiler to read include files only once, even if they are included more than one time with the `#include` preprocessor directive.

The `-notonce` option tells the compiler to read include files each time they are included with the `#include` directive.

The default is `-notonce`.

This option is equivalent to the `#pragma once` directive.

-pragma somepragma

There are a few situations where a legal pragma doesn't have a command line option. Use `-pragma` to pass a pragma to the top of each source file.

The precise position of the pragma will depend on whether the -pragma option was passed before `-d`, `-u`, `-prefix`, `-s` and `-once` options as these options are all "inserted" to the top of the source file in the order the options are passed.

The command line parser does not validate the pragma. You will get feedback on whether the syntax was correct if you have `-w` prag-

mas (on by default). Multi-word pragmas must be passed within quotes. For example `-pragma "export on"` will insert `#pragma export on` at the top of your file.

-precompile filename

Tells the compiler to precompile a header source code file and save the resulting file as `filename`.

For example, entering

```
MWC68K myfile.h -precompile myprecomp
```

will precompile the header file myfile.h and save the precompiled file as `myprecomp`.

Note that the file to be precompiled cannot contain any statements that generate data or object code.

If you use this option and you have code like this

```
#pragma precompile_target
```

in your source code you, will receive a warning. The warning tells you that the compiler is ignoring the pragma.

Mac OS

Unless the file name has a proper C++ naming extension, such as `.cp` or `.c++`, you must use the `-dialect cplus` option when creating a precompiled header for C++.

See also `-prefix filename`, `-load filename`.

-prefix filename

Tells the compiler to include the file `filename` at the beginning of the source code files specified in the command line.

Use this option to include often-used header files or precompiled header files.

See also `-load filename`.

-u name

BeOS

Undefine a preprocessing symbol.

-U name

BeOS

Undefine a preprocessing symbol.

Language Control

-ansi off/on/relaxed/strict

The `-ansi` options control ANSI C/C++ conformance.

The default is `-ansi off`. This allows you to use Apple-specific keywords, does not require ANSI C/C++ conformance, and sets the `-enum` option to `min`. Use this option when you write Macintosh code. On BeOS `-enum int` should be specified instead.

"On" and "relaxed" are synonymous. With this option, keywords for Mac OS programming—the `far`, `pascal`, and `asm` keywords—are *not* accepted. Only ANSI keywords are accepted. This option also turns on strict ANSI C/C++ conformance. This option works differently depending on whether your source code files contain C or C++ code. For C source code this option disallows:

- new style '//' comments
- non-`int` bitfields
- unnamed arguments in function definitions, such as:

```
void f(int)
{
...
}
```

- For C++ source code, this option disallows:

- empty array struct members,
- '#' tokens in a macro definitions that are not followed by a parameter,
- identifier token after `#endif`.

The `-ansi strict` option adds one more limitation to those specified for `-ansi on`. Enumerated types are always of type `int`. This additional restriction is the same as the `-enum int` option.

This option may also be manipulated with the `#pragma ANSI_strict` directive.

Mac OS

See also `-strict on/off, -enum int/min`

BeOS

Unless you pass other options for BeOS the defaults are -ansi off -enum int (these are order dependent).

-appleext on/off

Mac OS and BeOS

On tells the Mac OS compiler to recognize Apple's extended keywords: `far`, `asm`, and `pascal`. These keywords are not defined in the ANSI C and ANSI C++ standards. On the BeOS, specifying on allows the use of the `inherited` keyword.

Off tells the compiler to signal an error if it reads Apple's extended keywords: `far`, `asm`, and `pascal`. These keywords are not defined in the ANSI C and ANSI C++ standards.

The default is `-appleext on`.

-ARM on/off

Mac OS only

On tells the compiler to enforce some C++ language specifications described in the *Annotated C++ Reference Manual*, Ellis and Strous-

trup (ARM) rather than enforcing some conflicting ANSI C++ specifications. This disables certain ANSI C++ features that conflict with the ARM C++ specification. This is mainly used for compatibility testing.

Off tells the compiler not to enforce some C++ language specifications described in the *Annotated C++ Reference Manual*, Ellis and Stroustrup (ARM) that conflict with the ANSI C++ specifications.

The default is `-ARM off`.

This option is equivalent to the `#pragma ARM_conform` directive.

-bool on/off

Mac OS and BeOS

The compilers now allow for the use of the new C++ type `bool` in source code. The C++ language standard is evolving to allow the use of this new type. An example of how to use this new type is shown here:

```
bool foobar( short NewVal )
{
  bool myVar = true;
  if ( NewVal != 0 ) myVar = false;
  return myVar;
}
```

The default is `-bool off`.

This option is equivalent to the `#pragma bool` directive.

-c

Mac OS

Tells the compiler to check the syntax of the source file only and report any compiler errors. With this option, the compiler does not generate an object file.

BeOS

When this option is passed to the compiler, it means to compile the target file without automatically invoking the linker.

See also `-o name`, `-e`

-char signed/unsigned

Mac OS and BeOS

Treat unspecified characters as signed or unsigned. The default is signed.

This option may also be manipulated with the `#pragma unsigned char` directive.

-dialect c/cplus

Mac OS and BeOS

This option instructs the compiler whether to compile code as C or C++ code.

If this option is not invoked in any way, source files ending with the extension .c are compiled as C language files. Source files ending with .cp, .cpp, .cc, or .c++ are compiled as C++ language files.

The `-dialect c` option behaves the same way. It does *not* force C++ files to be compiled by the C compiler (an effort likely to fail in any event).

If `-dialect cplus` is invoked, files ending with .c are compiled as C++ language files.

This option may also be manipulated with the `#pragma cplus-plus` directive.

-ec++ onloff

Embedded

Enable Embedded C++. Disables some C++ features such as templates and exceptions; corresponds to a'C++ Language' panel setting in the CodeWarrior IDE. Default is off.

-EOL

This option enables the compilation of UNIX/DOS files. This option is off by default.

-j0

Mac OS

Pass `-j0` if you want to have multibyte characters in string literals and comments in your source.

-j1

Mac OS

Equivalent to `-j0`.

-j2

Mac OS

Equivalent to `-j0`.

-[no]mapcr

Mac OS

The `-mapcr` option tells the compiler to treat `\n` and `\r` character escape sequences as they are treated by the Apple MPW C compiler.

With the `-mapcr` option, the `\n` character is equivalent to a carriage return character (ASCII value 13) and the `\r` character is equivalent to a line feed character (ASCII value 10).

The `-nomapcr` option tells the compiler to treat `\n` and `\r` character escape sequences as they are treated in UNIX™ and in regular Macintosh text files.

With the `-nomapcr` option, the `\n` character is equivalent to the line feed character (ASCII value 10) and the `\r` character is equivalent to the carriage return character (ASCII value 13).

The default is `-mapcr`.

This option is equivalent to the `#pragma mpwc_newline` directive.

-mpwc

MWC68K only.

Generate code using MPW calling conventions.

This option is equivalent to the `#pragma mpwc` directive.

-objc onloff

Enable Objective-C language extensions; corresponds to a 'C++ Language' panel setting in the CodeWarrior IDE. The default is off.

-relax_pointers

Mac OS

Same as `-mpw_chars, and also the same as the -mpw_pointers option`.

Relaxes the rules for `char` pointers (`char`, `unsigned char`, and `Ptr`). This option is especially useful if you are working with code written before the ANSI C Standard. This old code used these types interchangeably. Use this option when converting MPW C source code to Metrowerks C.

This option does not work correctly when used with C++ source code because of the strong type checking enforced by C++.

This option is equivalent to the `#pragma mpwc_relax` directive.

-msext on/off

This option is equivalent to
```
#pragma cpp_extensions
```

Refer to the *C, C++, and Assembly Language Manual* for more information.

The default is `-msext off`.

-proto auto/strict/ignoreoldstyle

Tell the compiler whether to require prototypes. The default value is "auto" which does not require prototypes. The "strict" option is the same as `-r[equireprotos]`. The `-proto ignoreoldstyle` option tells the compiler to ignore old-style C function prototyping.

-r[equireprotos]

Tells the compiler to require ANSI-style function prototypes when compiling C and C++ source code. Default is `-proto auto`, which means that prototypes are not required. `-warnings all` turns on this option.

See also `-w[arnings] option[,option...]`.

This option is equivalent to the `#pragma require_prototypes` directive.

-RTTI on/off

Mac OS and BeOS

Enables run-time type information for C++ code. RTTI is enabled by default for BeOS and disabled by default for Mac OS.

This option may be manipulated with the `#pragma RTTI` directive.

-strict on/off

This option enforces a strict interpretation of the ANSI standard. In other words, language-related restrictions are put in place as detailed in `-ansi off/on/relaxed/strict`.

The default is `-strict off`.

-trigraphs on/off

Trigraph sequences are special codes that signify a single character. Trigraph sequences are meant to represent characters that are not on some non-English computer keyboards.

An ANSI trigraph sequence is two question mark characters followed by a third character. For example, use `??\` to represent a backslash, `\`.

On tells the compiler to recognize ANSI trigraph sequences.

Off tells the compiler not to recognize ANSI trigraph sequences.

The default is `-trigraphs off`.

This option may be manipulated with the `#pragma trigraphs` directive.

-wchar_t on/off

Enable `wchar_t` as a built-in C++ type, corresponds to the 'C++ Language' panel setting in the CodeWarrior IDE. The default is on.

Input File Control

NOTE: To avoid typographic confusion with the colon punctuation mark, the text in this section uses the term “MPW’s default directory” to refer to “`:`” in MPW. All examples, of course, do use “`:`”.

In most situations, the default behavior of the CodeWarrior command line tools is what you want. After a brief discussion of what constitutes default behavior, we will cover how you can change the behavior to suit your needs.

The CW command line compilers require that you expressly enter on the command line or in the makefile the directories you want searched for include files (headers). The search directories are not searched recursively. For example

```
MWCPPC foo.c -i sources
```

There are two directories that are usually added for you if you are using MPW: MPW's current working directory (:) and the folder "{MWCIncludes}". The previous example is interpreted by the compiler as:

```
MWCPPC foo.c -i : -i sources -i "{MWCIncludes}"
```

The include directive takes two forms. Assume that `foo.c` has the following:

```
/*
*   foo.c
*/

#include <types.h>
#include "foo.h"
```

When the header file is enclosed in angle brackets (<>), only system include directories are searched. When the header file is enclosed in double quotes (""), the implicit current working directory is searched first, then directories you pass with `-i`, and if still not successful, then the system includes "{MWCIncludes}".

In this example, `types.h` is found only if it exists in "{MWCIncludes}". If a file named `types.h` exists in the sources directory, it will not be found. However, the file `foo.h` can be located in MPW's default directory (:), sources, or "{MWCIncludes}".

The first `foo.h` that is found will be the one used. On the Macintosh, the header file names are case insensitive: `Foo.h` equals `foo.h`.

When you pass more than one include directory on the command line, they are searched in the order they are passed. The CW compilers do support a shorthand so that you can pass the include directories in a comma separated list.

```
MWCPPC foo.c -i sources,utilities,core_engine
```

The CW command line compilers also allow you to have UNIX-style include directives such as

```
#include <sys/stat.h>.
```

The CW command line compilers will search for that case first. Then, they will search all of the include directories for `sys/stat.h`. If `sys/stat.h` is still not found, then the compiler will internally convert the name to `:sys:stat.h` and try again.

Full and partial paths are supported for Mac and UNIX file naming conventions.

Using `.` or `..` in the directive are not supported.

Note that the MPW environment uses a colon ":" as the directory delimiter, while the BeOS development environment uses a forward slash "/."

-c

BeOS

mwcc only.

This option suppresses invocation of the mwld linker.

-cwd

Some compilers from a UNIX background do not use the current working directory as the first directory to search. Instead, they search the directory of the current source file first. You can duplicate this behavior with `-cwd source`.

```
MWCPPC foo.c -i sources -cwd source
```

translates to (assuming foo.c is in the folder FooFolder)

```
MWCPPC foo.c ∂
  -i FooFolder,sources,"{MWCIncludes}"
```

When you use `-cwd source`, please note that MPW's default directory (:) is not in the search path at all.

The `-cwd proj` option is equal to the default behavior; the default directory is always searched first.

The `-cwd explicit` option tells the compiler to not search in the default directory or `FooFolder`, but to begin the search in the first explicitly passed directory. For example

```
MWCPPC foo.c -i sources -cwd explicit
```

would translate to

```
MWCPPC foo.c -i sources,"{MWCIncludes}"
```

-i directory

Tells the compiler to search directory for files specified with the `#include` directive.

See also `-nosyspath`.

-I directory

BeOS

mwcc only.

Tells the compiler to search directory for files specified with the `#include` directive.

-i-

Mac OS and BeOS

Imagine that Apple comes out with some new technology. Because the technology is evolving, Apple doesn't want to put the headers into the Universal Headers (or "{MWCIncludes}"), so the header files appear in a separate folder. You have a couple of ways to work with this.

If you want all of your makefiles to have access to these new headers, (let's call the folder WizzBangStuff), then you can redefine "{MWCIncludes}". If you are using MPW, in UserStartup•Metrowerks, change the Set command to

```
# {MWCIncludes} - Directories to search for
MetroWerks C Include files
Set MWCIncludes ∂
"MacintoshHD:Development:WizzBangStuff,{MPW}Inte
rfaces:MWCIncludes:"
Export MWCIncludes
```

If you don't want all of your projects to use the headers in WizzBangStuff, but you want to access them with <>, then in your makefile use the `-i-` option. This option creates a delimiter to distinguish between your own (or user) directories and system directories. Once the CW compilers see a `-i-` on the command line, all remaining `-i` options will be treated as system directories. If the compiler encounters <>, it will look in those directories before looking in "{MWCIncludes}".

In addition, `-i-` does one other thing; it implicitly passes `-cwd` for you. (This is a UNIX convention.)

```
MWCPPC foo.c -i sources -i- -i WizzBangStuff
```

would translate to

```
MWCPPC foo.c ∂
  -i sources,WizzBangStuff,"{MWCIncludes}"
```

where WizzBangStuff will be the first directory to search for <>.

If you do want either the current working directory or the source file's directory searched first, you can pass either -cwd proj or -cwd source *after* the `-i-` option.

```
MWCPPC foo.c ∂
  -i sources -i- -i WizzBangStuff -cwd proj
```

would translate to

```
MWCPPC foo.c ∂
  -i :,sources,WizzBangStuff,"{MWCIncludes}"
```

-nodefaults and -nostdinc

On the Macintosh, these options are equivalent with respect to the compilers; "{MWCIncludes}" will not be implicitly added to the search path. This allows you to use new versions of Universal Headers, for example. On the BeOS, the defaults are defined by the system variable $BEINCLUDES in the profile file.

```
MWCPPC foo.c -i sources -i- ∂
  -i NewUniversalHeaders -cwd proj -nodefaults
```

would translate to

```
MWCPPC foo.c -i :,sources,NewUniversalHeaders
```

where NewUniversalHeaders will be the directory to search for <>.

If you need control over which folders are searched, leave off the -cwd proj option.

```
MWCPPC foo.c -i sources -i- ∂
  -i NewUniversalHeaders -nodefaults
```

would translate to

```
MWCPPC foo.c -i sources,NewUniversalHeaders
```

where NewUniversalHeaders will be the directory to search for <>.

-nosyspath

Tells the compiler to treat `#include <header>` directives as if they are `#include "header"` directives. In other words, “system” include files (`#include <header>`) are searched for in the same way as normal include files (`#include "header"`) are searched for.

Not all compilers are so rigorous that they only search the system directories when they encounter angle brackets. As a result, there is a lot of source code out there that has include directives with <> where they really should be "". In those situations you can use `-nosyspath`. `-nosyspath` forces the compiler to treat all <> as "".

There is a pitfall you will want to watch out for; if you have a types.h in your current working directory and in "{MWCIncludes}", the former (first one found) will be used.

On the BeOS, the default is to only use $BEINCLUDES for <>.

See also `-i directory`, discussed in the section entitled "Include file searching."

Diagnostic Message Control

-help

This option prints a list of the command-line options for the compiler tool to standard output.

-jm

Generates a warning and invokes the `-mpw_chars` option.

-maxerrors number

Sets the number of errors the compiler issues before aborting the compilation. If the compiler generates *number* errors, it stops compiling.

The default is `-maxerrors 51`.

-make

BeOS

Create a file dependency list only and send it to stdout.

-nofail

BeOS

On multiple source compiles don't quit compiling if there are errors in one or more files.

-p

Tells the compiler to write progress information to diagnostic output as it compiles.

-t

Tells the compiler to write compilation time to diagnostic output. The default is to not report compilation time.

-version

This option prints the version of the compiler tool on standard output.

-woption

BeOS

Table 5.2 Possible Usages of the -w BeOS Compiler Option

Command Line Option	Description
-w0	turn off all warnings
-w1	turn on all warnings except for command line options
-w2 through -w8	turn on all warnings except for command line options
-w9	turn on all warnings plus require prototypes

This option may be manipulated with the `#pragma warn_` family of directives. By default, warning messages are off for the BeOS.

-w[arnings] option[,option...]

Tells the compiler to turn compiler warnings on or off as instructed by the options. The options are stated in a comma-separated list

from the options listed below. The default is `-w[arnings] on`, except for BeOS, which defaults to `off`. See also `-strict on/off`.

- `on`—turn on all warnings (except `iserror`).
- `all`—turn on all warnings (except `iserror`) and require prototypes.
- `full`—same as `all`.
- `off`—do not generate any compiler warnings.
- `[is]err[or]`—treat warnings as errors. Note that the compiler does not generate object code if errors are generated.
- `[no]emptydecl`—generate a warning of an empty `typedef`, `struct`, or `class` (C++ only) declaration. The source code in Listing 1.1 will generate a warning.
- `[no]extended`—generate a warning when compiler encounters
  - illegal C++ enum conventions in ANSI C source code.
  - non-void functions that do not return a value (Listing 5.2).
- `[no]extracomma`— generate a warning if the compiler finds a comma after the last item in an enumerated list (Listing 5.3).
- `[no]hidevirtual`—warn on any hidden virtual functions. The default for this option is on.
- `[no]params`—generate a warning when there are unsupported or substituted command-line options.
- `[no]possible`—generate a warning if it reaches these possible errors:
  - a semicolon found immediately following a `while`, `if`, or `for` statement that may cause an unintended logical error
  - an assignment within a logical expression in a `while`, `if`, or `for` statement that may be meant as an equality operation—for example `if(temp=6)` instead of `if(temp==6)`
  - when the compiler believes it finds an unwanted comparison.
- `[no]pragmas`—generate a warning if it reaches an unrecognized `#pragma` directive.

- `[no]unusedarg`— generate a warning if it encounters a function argument that is not used in the function (Listing 5.4).
- `[no]unusedvar`—generate a warning if it encounters a variable that is declared but not used in a statement block (Listing 5.5).
- largeargs—will issue a warning if passing a float, double, or long long (64-bit) to an unprototyped function using Kernighan and Ritchie-style C code. This is helpful for finding suspect areas in old C code without requiring prototypes. This is for the BeOS only.

This option may be manipulated with the `#pragma warn_` family of directives.

Listing 5.1 Empty declaration warning

```
...
typedef int;// An empty declaration warning
...
```

Listing 5.2 Extended warning

```
...
long DivBy2(long a)
{
  a /= 2;
} // warning: does not return a long value
```

Listing 5.3 Extra commas in enumerations

```
...
enum test {
  apples,
  oranges,
  pears,
  bananas,// Generates an extra comma warning
```

```
};
...
```

Listing 5.4 Unused arguments

```
void test(int i, char b)
{     // Generates a warning because b is not used.
  a = 10;
}
```

Listing 5.5 Unused variables

```
void main(void)
{
  int i; // Generates warning because i not used.
  float x;

  x = 1.238343;
}
```

-w [no]implicitconv

Warn when different types are implicitly converted, corresponds to a 'C++ Warnings' panel setting in the CodeWarrior IDE.

-w [no]notinlined

Warn when the compiler doesn't inline inline functions. This option corresponds to a 'C++ Warnings' panel setting in the CodeWarrior IDE.

Debugging Control

-g

BeOS

This option causes the generation of symbolic debugging information. This option is passed to the mwld linker. This option is equivalent to -sym full.

This option may be manipulated with the `#pragma sym` directive.

-mbg on/off/full/ch8

MWC68K only.

Tells the compiler whether to generate MacsBug debugging symbols. The `-mbg full` option also turns on A6 stack frames with the `-a6` option. The `-ch8` option tells the compiler to generate old-style MacsBug names, which use the first eight characters of the name.

The default is `-mbg full`.

This option may be manipulated with the `#pragma macsbug` and `#pragma oldstyle_symbols` directives.

Mac OS

See also `-[no]a6`, `-profile on/off`.

-sym on/off/full/fullpath

Tells the compiler whether or not to generate debugging information. The `-sym on` option is *not* the same as the `-sym full` or `-sym fullpath` options. The `-sym full` and `-sym fullpath` options are the same. They tell the compiler to generate debugging information including full path names.

When generating 68K object code to be debugged with the CodeWarrior debugger, make sure the `-a6` option is specified with

the `-sym full` or `-sym on` options so that stack frames are always generated.

This option may be manipulated with the `#pragma sym` directive.

Mac OS

See also `-[no]a6`, `-profile on/off`.

-sym traceback

Mac OS

MWCPPC only. Same as `-traceback`.

This option may be manipulated with the `#pragma traceback` directive.

-traceback

Mac OS and BeOS

MWCPPC and mwcc only.

Generates AIX traceback tables. The default is off.

Profiling Control

-profile on/off

Mac OS and BeOS

On tells the compiler to generate object to be used with the CodeWarrior Profiler.

Off tells the compiler not to generate object to be used with the CodeWarrior Profiler. For information on the libraries that must be linked with your program to use the profiler, see the *CodeWarrior Profiler* manual.

The default is `-profile off`.

This option may be manipulated with the `#pragma profile` directive.

Debugging is easier with profiling off. The CodeWarrior Profiler requires routine entry and exit configurations that interfere with the debugger's ability to easily follow program execution. See the *CodeWarrior Profiler* manual for more information on debugging profiled code.

Currently there is no profiler library for BeOS, but one may become available in the future.

6

Metrowerks Pascal

This chapter lists and describes the Pascal language compiler options. Options control the way the compiler translates source code, generates object code, and communicates with the programmer.

Overview of Metrowerks Pascal

The Metrowerks Pascal command line compilers —MWPascalPPC (Mac OS) and MWPascal68K (Mac OS)—translate Pascal source code into PowerPC or 68K object code. Object code generated by these compilers is processed by the Metrowerks command-line linkers—MWLinkPPC (Mac OS) and MWLink68K (Mac OS)—to produce executable objects like applications, code resources, and libraries.

Each compiler has various options that you can set. If you are familiar with the CodeWarrior development environment you will see that many of these options correspond to settings in the CodeWarrior Preferences dialog. Refer to "Comparison Overview" to learn about the relationships between the command-line tools and the CodeWarrior IDE.

This chapter organizes options into these groups:

- Code Generation Control—options that affect how object code is created and arranged in the compiler's output file.
- Preprocessing and Precompiling—options that affect preprocessor macro definition settings and precompiled file control.
- Language Control—options that affect how the compiler translates source code.
- Input File Control—options that affect file inclusion.
- Diagnostic Message Control—options that determine how the compiler communicates warnings, errors, and progress information to the user.

- Debugging Control—options that tell the compiler to generate object code for use with debuggers.
- Profiling Control—options that tell the compiler to generate object code for use with the CodeWarrior Profiler.

In all cases, text in braces—the [and] characters—is optional. The braces themselves never appear in the actual command. For example, you could write `-str[ings] pool` as either `-strings pool` or `-str pool`.

Where an option has several possible permutations, the possibilities are separated by a slash—the / character. For example, `-sym on/off/full/fullpath` reflects the fact that the `-sym` command may be followed by one or more of four possibilities. So you might have `-sym on`, `-sym off`, or `-sym full`.

In cases where you provide a variable parameter, like a file name, that item is in italic text. For example, `-precompile` *filename* means you must provide a file name. The explanation that accompanies each compiler option makes it clear what you must provide.

You can learn how to implement many of the compiler options documented here using compiler directives ($) by referring to the *Pascal Language Reference* manual on the CodeWarrior Reference CD.

Code Generation Control

-a4

Mac OS

For MWPascal68K only.

Tells the compiler whether to use the 68K processor's A4 register for global data references.

Use this option when creating 68K code resources.

See also `-coderesource` on page 14.

-[no]a6

For MWPascal68K only.

Tells the compiler whether to generate function stack frames based on the A6 register. The default is `-a6`.

With A6 stack frame generation off, `-noa6`, the compiler does not generate stack frames when they are unnecessary. This results in slightly smaller, faster object code that is harder to trace in a low-level debugger.

Mac OS

Use the `-a6` option when generating object code that will be used with the CodeWarrior debugger.

-align mac68k/mac68k4byte/power

This option specifies alignment for data structures. With `mac68K`, the compiler aligns an item on a 2-byte boundary, unless it is only 1 byte long. The `-align mac68k4byte` option specifies 68000-family 4-byte alignment for data structures.

This option is equivalent to the `$ALIGN` directive.

Mac OS

The `power` option specifies PowerPC alignment for data structures. This option aligns data on its natural boundary. In other words, a character is aligned on a 1-byte boundary, a 16-bit integer is aligned on a 2-byte boundary, and so on. This alignment applies recursively to structured data and arrays containing structured data. So, for example, an array of structured types containing an 8-byte floating point member must be aligned on an 8-byte boundary.

The default for MWPascalPPC is `-align power`. The default for MWPascal68K is `-align mac68k`.

-b

For MWPascal68K only.

Tells the compiler whether to put string constants into instruction code instead of data. This option also tells the compiler to generate references to string constants and functions relative to the 680x0 processor's program counter (PC register) if they are in the same segment.

The default is to generate references relative to the A4 or A5 registers of the 680x0 processor (depending on the use of the `-a4` option).

See also `-a4`.

-b2

For MWPascal68K only.

Has the same functionality as the `-b` option. In addition it places identical string constants in the same memory space.

See also `-b`.

-d0

MWPascal68K only.

Tells the compiler to return pointer types in the D0 register of the 680x0 processor instead of the A0 register.

Mac OS

Use this option when generating object code that deals with the MPW Shell or MPW libraries.

-dsize 64/80

For MWPascal68K only.

Specify the size of the double data type. You specify the number of bits. To specify 8-byte `doubles`, use `-dsize 64`. To specify 10-byte `doubles`, use `-dsize 80`. The default is `-dsize 64`. 10-byte `double` size is the same size used by the SANE `extended` data type.

-ext extension

Specifies the filename extension of the compiler's output file.

If *extension* has a leading period (`.`) then *extension* is added to the end of the name of the compiler's output file, which contains object code. If *extension* has no leading period then *extension* replaces the output filename's extension.

For example,

```
MWPascal68K SockSorter.p -ext .mw.p.o
```

specifies an output file named SockSorter.c.mw.p.o.

```
MWPascal68K SockSorter.p -ext lib
```

specifies an output file named SockSorter.lib.

By default, *Extension* is "`.o`". The `-ext` option is ignored if the `-o name` option is used.

-fatext

Mac OS

Changes the default output name extension. For MWPascalPPC, from .o to .ppc.o. For MWPascal68K, from .o to .68k.o. If an actual output file is passed with `-o name`, `-fatext` does nothing.

-force_active

Mac OS

For MWPascal68K only.

Tells the compiler to mark all code so that it will not be removed by the linker, even if the code is "dead."

-fp_contract on/off

Mac OS

For MWPascalPPC only.

Tells the compiler to enable or disable PowerPC floating-point contraction instructions (for example, `FMADD`, `FMSUB`). The default is `-fp_contract on`.

-ldsize 64

Mac OS

MWPascalPPC only.

Tells the compiler to use 8-byte doubles (64 bits). For PowerPC, only 8-byte doubles are currently supported.

-mc68020

Mac OS

MWPascal68K only.

Tells the compiler to generate instructions specifically for the Motorola 68020, 68030, and 68040 processors.

Using this option creates object code that cannot be run on 68000 processors. By default, MWPascal68K generates object code compatible with all Motorola 68000-family processors.

-mc68881

MWPascal68K only.

Tells the compiler to generate code for mathematical operations for a Motorola 68881 or Motorola 68882 math coprocessor instead of calling library routines.

By default, MWPascal68K compiles mathematical operations into library calls.

Mac OS

Using this option creates object code that cannot be run on 68K Macintosh computers that do not have math coprocessors.

-model farCode/nearCode/codeSmart

MWPascal68K only.

The "farCode" option tells the compiler to always generate 32-bit references for executable code, even if a 16-bit reference is possible.

The "nearCode" option tells the compiler to always generate 16-bit references for executable code. If a 32-bit reference is necessary, the linker will post an error.

The `-model codeSmart` option generates 16-bit references whenever possible. If an object is beyond a 16-bit range, the compiler generates a 32-bit reference.

The default is `-model nearCode`.

-model farData/nearData

MWPascal68K only.

The `-model farData` option removes the 64K limit on global data and stores global data in far data space instead of near data space. When used, there is no limit on global data. The generated code will also be slightly bigger and slower.

The `-model nearData` option tells the compiler to generate 16-bit references to data. This option limits global data to 64K

The default is `-model nearData`.

-model far/near

MWPascal68K only.

The `-model far` option tells the compiler to use `-model farCode`, `-model farData`, `-model farStrings`, and `-model farVTables`. The `-model near` option tells the compiler to use `-model nearCode`, `-model nearData`, `-model nearStrings`, and `-model nearVTables`.

The default is `-model near`.

-model farStrings/nearStrings

MWPascal68K only.

The `-model farStrings` option tells the compiler to generate 32-bit references to string literals.

The `-model nearStrings` option tells the compiler to generate 16-bit references to string literals. This option limits literal character string storage to 64K.

The default is `-model nearStrings`.

-o name

Tells the compiler the name of the file or directory in which to save object code in the CodeWarrior object file format for 68K or PowerPC. If this option is not specified, or if you provide a directory name, the default file name for the object code file is the source code filename with a ".o" extension.

See also `-c`.

-opt option[, ….]

The `-opt` option controls compiler optimizations. The available optimizations differ for MWPascal68K and MWPascalPPC. In either case you specify a list of optimization options from the possibilities listed below. Multiple options must be separated from each other by commas, with no white space. You must supply at least one option. The default for both compilers is `-opt on`. The effect of `-opt on` is slightly different, because the two compilers have different possible

optimizations. The MWPascal68K compiler does not have scheduling and level-related options.

For MWPascal68K, the possible optimizer options are:

- `on`—turns on all optimizations. This is equal to `-opt global,peep, speed`.
- `off`—turns off all optimizations.
- `all`—turns on all optimizations. This is the same as `-opt on`.
- `full`—same as `all`.
- [`no`]`cse`—This option is equivalent to `-opt global on/off`
- [`no`]`global`—tells the compiler to generate object code with or without global optimization and use register allocation.
- [`no`]`peep`—tells the compiler to generate object code with or without peephole optimization. Peephole optimization performs small, local optimizations to improve execution speed. Peephole optimization eliminates redundant compare instructions and improves branch sequences.
- `size/speed`—tells the compiler to generate object code that is optimized for a smaller size or greater speed. These cannot both be turned on, they are alternatives. The compiler optimizes for smaller object code size at the expense of slower execution. The compiler optimizes for faster execution speed at the expense of larger object code size. You always have either `size` or `speed` selected. The default is `speed`.

Mac OS

For MWPascalPPC, the possible optimizer options are:

- `on`—turns on all optimizations, level 1. This is equal to `-opt global, peep,schedule,l1,speed`.
- `off`—turns off all optimizations.
- `all`—turns on all optimizations, level 4. This is the same as `-opt global,peep,schedule,l4,speed`.
- `full`—same as `all`.

- [no]`global`—tells the compiler to generate object code with or without global optimization and use register allocation.
- [no]`peep`—tells the compiler to generate object code with or without peephole optimization. Peephole optimization performs small, local optimizations to improve execution speed. Peephole optimization eliminates redundant compare instructions and improves branch sequences.
- [no]`schedule`—tells the compiler to generate object code with or without instruction scheduling optimization. Instruction scheduling means that the compiler arranges certain sequences of machine language instructions to take advantage of the PowerPC's pipeline architecture. The `-schedule` option is the same as `-opt schedule601`.
- `schedule601`, `schedule603`, or `schedule604`—tunes scheduling for the PowerPC processor specified. The `schedule601` option is the best compromise for mixed PowerPC machines.
- `l1`, `l2`, `l3`, or `l4`— sets the global optimization level. `l1` offers the least optimization, while `l4` offers the most optimization.
- `size/speed`—tells the compiler to generate object code that is optimized for a smaller size or greater speed. These cannot both be turned on or off, they are alternatives. The compiler optimizes for smaller object code size at the expense of slower execution. The compiler optimizes for faster execution speed at the expense of larger object code size. You always have either `size` or `speed` selected. The default is `speed`.

-ov

This option causes code for overflow checking to be generated. By default, overflow-checking code is not generated.

-r

This options causes range-checking code to be suppressed. By default, range-checking code is generated.

-str[ings] dontreuse

Tells the compiler not to reuse identical character string literals.

Preprocessing and Precompiling

-d name [= [value]]

Tells the compiler to define a preprocessor symbol with the name *name*. The default value for the defined symbol is TRUE. Use the optional item, *value*, to specify a value other than TRUE.

For example, entering

```
-d debug_on=TRUE
```

is equivalent to placing a constant definition in the Pascal code assigning debug_on the value of TRUE.

Language Control

-ans

Mac OS

This option tells the Pascal compiler to use ANS language conformance standards.

Refer also to the discussion "-turbo."

-c

Mac OS

Tells the compiler to check the syntax of the source file only and report any compiler errors. With this option, the compiler does not generate an object file.

-cs

Mac OS

This option tells the Pascal compiler to respect case sensitive identifiers.

-[no]mf

Mac OS

Tells the compiler whether to use Finder memory (the Process Manager heap) when necessary. Use of the `-mf` switch will dramatically save memory in the MPW Shell's partition. The default is `-mf`.

The `-nomf` option tells the compiler to *never* use MultiFinder memory. The Shell's partition must hold all data.

-nilchk

Mac OS

This option causes the compiler to generate code that traps references to nil.

-param

Mac OS

This option causes the compiler to always copy value parameters.

-relax

Mac OS

This option causes the Pascal compiler to relax pointer compatibility rules, so that the compiler is less restrictive about pointer types.

-sets pool/dontreuse

Mac OS

This option causes the compiler to do special things with sets.

The `pool` parameter causes sets to be pooled.

The `dontreuse` parameter causes sets to be generated each time they are needed (not shared).

-str[ings] pool/dontreuse/dynamic

Mac OS

This option causes the compiler to do special things with strings.

The `pool` parameter causes strings to be pooled in one memory area.

The `dontreuse` parameter causes strings to be generated each time they are needed (not shared).

The `dynamic` parameter causes strings to be copied with the length byts.

-trapcase

Mac OS

This option causes the compiler to trap unmatched case statements.

-turbo

Mac OS

This option tells the compiler to use Turbo Pascal's Input/Output (I/O) model. The default is to use MPW's I/O model.

Refer also to the discussion of "-ans."

Input File Control

NOTE: To avoid typographic confusion with the colon punctuation mark, the text in this section uses the term “MPW’s default directory” to refer to “`:`” in MPW. All examples, of course, do use “`:`”.

In most situations, the default behavior of the CodeWarrior command line tools is what you want. After a brief discussion of what constitutes default behavior, we will cover how you can change the behavior to suit your needs.

The CW command line compilers require that you expressly enter on the command line or in the makefile the directories you want searched for include files. The search directories are not searched recursively. For example

```
MWPascal68K foo.p –i sources
```

There are two directories that are usually added for you if you are using MPW: MPW's current working directory (:) and the folder "{MWPInterfaces}". The previous example is interpreted by the compiler as:

```
MWPascal68K foo.p –i : –i sources –i
"{MWPInterfaces}"
```

When you pass more than one include directory on the command line, they are searched in the order they are passed. The CodeWarrior compilers do support a shorthand so that you can pass the include directories in a comma separated list.

```
MWPascal68K foo.p –i
sources,utilities,core_engine
```

Using `.` or `..` in the directive are not supported.

Note that the MPW environment uses a colon “:” as the directory delimiter.

-cwd

Some compilers from a UNIX background do not use the current working directory as the first directory to search. Instead, they search the directory of the current source file first. You can duplicate this behavior with `-cwd source`.

```
MWPascal68K foo.p -i sources -cwd source
```

translates to (assuming foo.c is in the folder FooFolder)

```
MWPascal68K foo.p ∂
  -i FooFolder,sources,"{MWPInterfaces}"
```

When you use `-cwd source`, please note that MPW's default directory (:) is not in the search path at all.

The `-cwd proj` option is equal to the default behavior; the default directory is always searched first.

The `-cwd explicit` option tells the compiler to not search in the default directory or `FooFolder`, but to begin the search in the first explicitly passed directory. For example

```
MWPascal68K foo.p -i sources -cwd explicit
```

would translate to

```
MWPascal68K foo.p -i sources,"{MWPInterfaces}"
```

-i directory

Tells the compiler to search *directory* for include files.

-i-

Mac OS

Imagine that Apple comes out with some new technology. Because the technology is evolving, Apple doesn't want to put the headers into the Interfaces (or "{`MWPInterfaces`}"), so the files appear in a separate folder. You have a couple of ways to work with this.

If you want all of your makefiles to have access to these new headers, (let's call the folder WizzBangStuff), then you can redefine

"{MWPInterfaces}". If you are using MPW, in UserStartup•Metrowerks, change the Set command to

```
# {MWPInterfaces} - Directories to search for
MetroWerks Pascal Interfaces files
Set MWPInterfaces ∂
"MacintoshHD:Development:WizzBangStuff,{MPW}Inte
rfaces:MWPInterfaces:"
Export MWPInterfaces
```

If you don't want all of your projects to use the headers in WizzBangStuff, but you want to access them with <>, then in your makefile use the `-i-` option. This option creates a delimiter to distinguish between your own (or user) directories and system directories. Once the CW compilers see a `-i-` on the command line, all remaining `-i` options will be treated as system directories. If the compiler encounters <>, it will look in those directories before looking in "{MWPInterfaces}".

In addition, `-i-` does one other thing; it implicitly passes `-cwd` for you. (This is a UNIX convention.)

```
MWPascal68K foo.p -i sources -i- -i
WizzBangStuff
```

would translate to

```
MWPascal68K foo.p ∂
  -i sources,WizzBangStuff,"{MWPInterfaces}"
```

where WizzBangStuff will be the first directory to search for <>.

If you do want either the current working directory or the source file's directory searched first, you can pass either -cwd proj or -cwd source *after* the `-i-` option.

```
MWPascal68K foo.p ∂
  -i sources -i- -i WizzBangStuff -cwd proj
```

would translate to

```
MWPascal68K foo.p ∂
  -i :,sources,WizzBangStuff,"{MWPInterfaces}"
```

Diagnostic Message Control

-e filename

Writes errors to file named *filename*.

-h

This option suppresses error messages regarding unsafe handles.

-maxerrors number

Sets the number of errors the compiler issues before aborting the compilation. If the compiler generates *number* errors, it stops compiling.

The default is `-maxerrors 51`.

-p

Tells the compiler to write progress information to diagnostic output as it compiles.

-rebuild

This options causes all symbol table resources to be rebuilt.

-t

Tells the compiler to write compilation time to diagnostic output. The default is to not report compilation time.

-w[arnings] option[,option...]

Tells the compiler to turn compiler warnings on or off as instructed by the options. The options are stated in a comma-separated list from the options listed below. The default is `-w[arnings] on`, so that warning messages are displayed.

- `on`—turn on all warnings (except `iserror)`. This is the default.
- `all`—turn on all warnings (except `iserror)` and require prototypes.
- `full`—same as `all.`
- `off`—do not generate any compiler warnings.
- `[no]gotos`— generate a warning if it encounters a possibly dangerous goto construct in the code.
- `[no]foridx`—generate a warning on for index assignments.
- `[no]pragmas`—generate a warning if an unknown pragma is encountered.
- `[no]strings`—generate a warning if a possible string mangling could occur.
- `[no]returns`—generate a warning if no returns are encountered in a function.
- `[no]routines`—generate a warning if undefined routines are detected.

Debugging Control

-mbg off/full/ch8/on

MWPascal68K only.

Tells the compiler whether to generate MacsBug debugging symbols. The `-ch8` option tells the compiler to generate old-style MacsBug names, which use the first eight characters of the name. Otherwise, new-style Macsbug symbols are generated.

`-mbg full` turns on a6 stack frames also.

The default is `-mbg on`.

See also -a6.

-clean

This option causes all symbol table resources to be erased.

-k directory

This option causes symbol table resources to be generated in *directory*.

-noload

This option suppresses the use and generation of symbol table resources. By default, symbol tables are used and generated.

-sym on/off/full/fullpath

Tells the compiler whether or not to generate debugging information. The `-sym on` option is *not* the same as the `-sym full` or `-sym fullpath` options. The `-sym full` and `-sym fullpath` options are the same. They tell the compiler to generate debugging information including full path names.

You may also specify the following options, separated by commas with no white space between them. For example:

```
-sym on,nolines,novars
```

- nolines–this option is under construction
- notypes–this option is under construction
- novars–this option is under construction

-sym traceback

Mac OS

MWPascalPPC only.

This option turns on traceback table generation for the PowerPC processor.

Profiling Control

-profile on/off

Mac OS

On tells the compiler to generate object to be used with the CodeWarrior Profiler.

Off tells the compiler not to generate object to be used with the CodeWarrior Profiler. For information on the libraries that must be linked with your program to use the profiler, see the *CodeWarrior Profiler* manual.

The default is `-profile off`.

Debugging is easier with profiling off. The CodeWarrior Profiler requires routine entry and exit configurations that interfere with the debugger's ability to easily follow program execution. See the *CodeWarrior Profiler* manual for more information on debugging profiled code.

7

Metrowerks Linkers

This chapter lists and describes linker options. Options control the way the linker connects object code into an executable.

Overview of Metrowerks Linkers

The Metrowerks C/C++ command-line compilers—MWCPPC, mwcc and MWC68K—translate C and C++ source code into PowerPC or 68K object code. Object code generated by these compilers is processed by the Metrowerks command-line linkers—MWLinkPPC, mwld and MWLink68K—to produce executable objects like applications, code resources, and shared libraries.

Linker options are organized into these groups:

- Object Code Control—options that affect how object code is organized and how it is linked with other object code, including Mac OS SIZE resources.
- Segment Control—organizing, naming, and controlling segments and attributes
- Code Resource Control—options that affect how object code is arranged and linked when creating Mac OS code resources for 68K and PowerPC
- Code Fragment Control—options that affect how object code is arranged and linked when creating a code fragment container—an application, code resource, or library—for the Mac OS Code Fragment Manager
- Output File Control—options that affect what kind of output file the linkers create and what the linkers put in the output file
- Linker Operation Control—options that determine how the linkers use the computer's disks, memory, and processor while running

- Diagnostic Message Control—options that affect how the linkers communicate warnings, errors, and progress information to the user
- Debugging Control—options that tell the linkers to generate an output file that can be used with debugging software, such as CodeWarrior Debug

In all cases, text in braces—the [and] characters—is optional. The braces themselves never appear in the actual command. For example, you could write `-str[ings] pool` as either `-strings pool` or `-str pool`.

Where an option has several possible permutations, the possibilities are separated by a slash—the / character. For example, `-sym on/off/full/fullpath` reflects the fact that the `-sym` command may be followed by one or more of four possibilities. So you might have `-sym on`, `-sym off`, `-sym full`, or `-sym fullpath`.

In cases where you provide a variable parameter, like a file name, that item is in italic text. For example, `-o` *filename* means you provide a file name. The explanation that accompanies each linker option makes it clear what you must provide.

Object Code Control

-collapsereloads

Mac OS

MWLinkPPC only.

Collapse unused TOC reload slots; removes unnecessary NOP instructions after function calls; works when linking applications and other kinds of code fragments; corresponds to a new 'PPC PEF' panel setting in the IDE.

-custom

Mac OS

MWLink68K only.

This option is used for code resources and applications. It requires that you write your own startup code.

Refer to the *Targeting Mac OS* manual for more information.

-dead on/off/code/initcode

This option controls deadstripping. This option tells the linker whether routines and variables not referenced by any other object files should be linked into the output file. The `-dead` option can distinguish between code in your object files and linker-generated initialization code.

The `-dead on` option tells the linker to not link either unused code or unused init code. On is the default. The `-dead on` option has no effect when building a library.

The `-dead off` option includes all code. The `-dead code` option tells the linker to not link unused code but does link init code. The `-dead initcode` option tells the linker to not link init code but does link object unused object code.

-intsize 2/4

MWLink68K only.

Specifies the size of the `int` data type, in bytes, to use when linking object code. To specify 2-byte `int`s, use `-intsize 2`. To specify 4-byte `int`s, use `-intsize 4`. The default is `-intsize 4`.

NOTE: The `int` size used for the linker should be the same as that used for the compiler.

Mac OS

When creating an MPW tool use `-intsize 4`.

-model near/far

MWLink68K only.

The `-model near` option specifies that the jump table entries are put in near data space. Use this option when using compiler options `-model nearData` and `-model nearCode`.

The "far" option specifies that the jump table entries are put in far data space. Use this option when using `-model farData` and `-model farCode`.

To learn morea bout this option, refer to the discussion in "-model farCode/nearCode/codeSmart."

The default is `-model near`.

-size on/off

Mac OS

Create a 'SIZE' resource for an application. The default is off. A 'SIZE' resource will not be made for code resources, libraries, or MPW tools regardless of this setting.

-sizemax num

Mac OS

Specify the maximum size of the heap with *num* in kilobytes.

Default is 384.

-sizemin num

Mac OS

Specify the minimum size of the heap with *num* in kilobytes.

Default is 384.

-sizeflags attribute

Mac OS

Attribute can be a hexadecimal value (preceded by a "$" character), a decimal value, or a comma-separated list of these attributes:

- `suspendevents`
- `canbackground`
- `multifinderaware`
- `onlybackgroud`
- `getfrontclicks`
- `childdiedevents`
- `32bitclean`
- `highlevelevents`
- `remoteevents`
- `stationary`
- `textedit`
- `displayaware`

Segment Control

-glue mainseg/curseg

Mac OS

MWLink68K only.

Allows you to control where the linker puts compiler generated code. The default is `-glue mainseg`. This puts the code in the main segment.

If you need to control the size of the main segment (for example, to use the near code model), you can use `-glue curseg` to put the compiler generated code into the current segment. Please note that linker generated code is now put into the first segment by default. In old versions of CodeWarrior it was put into the current segment.

-IDEseg[ments]

Mac OS

MWLink68K only.

Match names and IDs for segments created with `-sb` to those generated by the CodeWarrior IDE. The default is to have no gaps in the segment IDs and to allow segment remapping (`-sn`/`-sg`) of segments created with `-sb` instead of matching.

-ma name=alias

Mac OS

MWLink68K only.

Create an alias for the item *name*. For example, assume you have a library that contains a function named `myFunc()`, but your code always calls it using upper case, `MYFUNC()`. The linker will never find `MYFUNC()`, and the build will fail. You can use `-ma MYFUNC=myFunc` to create an alias for the name. If the linker can't find a definition for `MYFUNC()`, it looks in a list for aliases. If it finds one—`myFunc` in this case—the linker then tries to find a definition for `myFunc()`.

NOTE: If the linker does find `MYFUNC()`, it never looks for `myFunc()`, so be careful about name collisions. Also, there is no change in the code. If `MYFUNC()` can't be found, the reference is

still to `MYFUNC()`. However, everytime the code attempts to call `MYFUNC()`, `myFunc()` is called instead.

-ra [segment]=attribute[,...]

Sets code segment name and attributes.

The segment name may be 255 characters long. If no segment name, *segment*, is specified, then this option specifies the default code segment attributes for the remaining segments to be linked.

The segment name, *segment*, is the same segment specified in `#pragma segment` directives in the source code or the `-s name` option used with the MWC68K and MWLink68K tools.

Specifying the segment name replaces the default attributes.

Attribute can be a hexadecimal value (preceded by a "$" character), a decimal value, or a comma-separated list of these attributes:

- `resSysHeap`
- `resPurgeable`
- `resLocked`
- `resProtected`
- `resPreload`

This option has no effect on segment 0 (the jump table) and segment 1 (the main code resource).

Mac OS

When creating a code resource, use the `-rsrcflags list` option.

The MWLink68K MPW tool handles segment names in code resources differently than the traditional Apple linkers. MWLink68K has two options, modeled after the CodeWarrior IDE, that allow you to control the name and attributes of the first or main segment of a code resource: `-rsrcname` name and `-rsrcflags` flags. In the IDE the `-rsrcflags` equivalent is rather restrictive since it doesn't allow you to change the attributes of any segment other than the

main segment. Use the `-ra` to change the attributes of any named segment.

Mixing the options `-rsrcname` and `-rsrcflags` with `-ra` can lead to unexpected results. Consider the following example:

```
-rt INIT=45 -ra Main=resLocked -rsrcname "MyINIT"
```

You might think that your first segment would be named `MyINIT` with attribute `resLocked`. Instead, MWLink68K changes the main segment's name to `MyINIT` before it sets the attributes and since there are no attributes specified for a segment named `MyINIT`, the attributes get set to 0. As far as the linker is concerned

```
-ra Main=resLocked
```

is setting the attributes of a segment that doesn't exist. There are several ways to fix this ambiguity. The following examples each will create a locked main segment.

```
-rt INIT=45 -rsrcflags locked -rsrcname "MyINIT"
-rt INIT=45 -ra Main=resLocked -sn Main=MyINIT
-rt INIT=45 -ra MyINIT=resLocked -rsrcname"MyINIT"
```

Of the three examples, the first and second are the most straightforward. They either completely use MWLink68K options or Apple's Link options.

- If you are building a multi-segment Desk Accessories or 'DRVR' code resources, you must use the MWLink68K options (first example) exclusively to set the name of the main segment.
- The only disadvantage to the second example is that it won't work in all cases, as when building multi-segment Desk Accessories or 'DRVR' code resources.
- The third example mixes options but will still work.

Regardless of which type of code resource you are building, `-ra` is the preferred method of changing the attributes of all segments ex-

cept the main segment. See "Segmenting 68K Code" on page 5 for more information.

-rn

MWLink68K only.

Tell the linker to not include resource (segment) names in the output file.

-s name

MWLink68K only.

This is synonymous with the `-sb` option, described below.

See also `-fastlink on/off`.

-sb name [=attribute [,...]]

MWLink68K only.

Creates a new code segment named *name* using the next object code file specified in the MWLink68K parameter list. The name may be up to 255 characters long. Use this option for file-level segmentation of your program's object code files instead of function-level segmentation specified using the `#pragma segment` directive in source code files.

Object files listed after this option are added to the segment named *name*. To rename the main segment, "`Main`," place a `-sb` option before the first object file that is specified.

You may also specify resource attributes for the segment. For information on attributes, see "-ra [segment]=attribute[,...]."

For more information, see "Overview of Segmenting 68K Code".

-sf filename.seg

MWLink68K only.

This option allows you to pass a .seg file used with a CodeWarrior IDE project to the CodeWarrior MPW linker. You may pass `-sf filename.seg` with different .seg files and use them in conjunction with `-sg` and `-sn` options. The effect is cumulative.

-sg newSegment = [oldSegment [,...]]

MWLink68K only.

Creates a new segment, *newSegment*, by merging existing segments. Existing segments are specified in a comma-separated list.

If the segment list, *oldSegment, …,* list is not specified, then all code segments are linked into a single segment named *newSegment*. The name may be up to 255 characters long.

-sn oldseg=newSeg

MWLink68K only.

Merge segment *oldseg* into segment *newSeg*. The segment names may be up to 255 characters long.

-single

MWLink68K only.

Creates the executable as one code segment.

-tmp directory

MWLink68K and MWLinkPPC only.

Creates temporary files and put them in the specified directory during the linking process. This may make it easier to link very large projects in limited RAM. The default is to not use temp files.

For MWLink68K, this option is more effective with `-fastlink off`. Temp files are created for PEF and 'obj ' files only.

For MWLinkPPC, this option is more effective with `-link slow`. Temp files are created for PEF and 'XCOF ' files only.

If this option is used, it is most effective when you have many of these files.

The directory you specify is created if it doesn't exist. You can use MPW's environment variable "{TempFolder}" for the directory.

Code Resource Control

-coderesource

Mac OS

Same as `-xm c[oderesource]`.

-custom

Mac OS

MWLink68K only.

Tells the linker to use a custom code resource header if you also pass `-xm c[oderesource]`. This option no longer turns on the `-xm c[oderesource]` option automatically.

See the CodeWarrior *Targeting Mac OS* manual for more information.

-da

Mac OS

MWLink68K only.

When creating a code resource, tells the linker to use a Desk Accessory code resource header. If you use `-da`, the linker will automatically add a null character at the beginning of the first segment name.

This option also turns on the `-xm c[oderesource]` option.

-drvr

Mac OS

MWLink68K only.

When creating a code resource, tells the linker to use a driver code resource header.

This option also turns on the `-xm c[oderesource]` option.

-fat filename

Mac OS

MWLinkPPC only.

When creating a code resource, tells the linker to use a fat code resource header. *Filename* can be any file that contains a 68K code resource. The linker will look in *filename* to find a resource with the type and ID that match the type and ID of the current PowerPC code resource, and use that resource to form a fat resource.

This option also turns on the `-xm c[oderesource]` option.

See also `-safe filename`.

-merge on/off

Mac OS

The `-merge on` option combines the newly linked resources with the existing output file instead of creating a new output file. The `-merge off` option wipes clean both the resource and data forks. If no file type or file creator is specified with the `-c creator` and `-t type` options, the existing creator and type are used.

The default is `-merge on`.

See also `-rsrcmerge`.

-native

Mac OS

MWLinkPPC only.

When creating a code resource, tells the linker to use a native code resource header.

This option also turns on the `-xm c[oderesource]` option.

See also `-fat filename`, `-safe filename`

-rsrcflags list

Mac OS

Sets the resource flags on the generated resources. *List* is a comma-separated list (no white space) of any combination of

- `system`
- `purgeable`
- `locked`
- `protected`
- `preload`

See also `-ra [segment ]=attribute[,...]`, `-rsrcmerge`, `-rsrcname name`.

-rsrcmerge

Mac OS

Same as `-merge on`.

See also `-merge on/off`, `-rsrcflags list`, `-rsrcname name`

-rsrcfar

Mac OS

MWLink68K only.

Use this option when creating a code resource that uses far addressing.

This option also turns on the `-xm c[oderesource]` option.

See also `-model near/far`.

-rsrcmulti

Mac OS

MWLink68K only.

Same as `-rsrcfar`.

See also `-model near/far`, `-rsrcfar`, `-rsrcvirfuncs`.

-rsrcname name

Mac OS

Sets the resulting code resource's name as *name*. The name may be up to 255 characters long.

This option also turns on the `-xm c[oderesource]` option.

-rsrcsegtype type

Mac OS

MWLink68K only.

Tells the linker to set the resource type to give to all segments except the main segment in a multi-segment code resource. The resource type of the main segment of the code resource is specified with `-rt`.

This option also turns on the `-xm c[oderesource]` option.

-rsrcvirfuncs

Mac OS

MWLink68K only.

Same as `-rsrcfar`. Use this option when creating a C++ code resource. This option makes sure virtual function tables are correctly linked.

See also `-model near/far`, `-rsrcfar`.

-rt type='id'

Mac OS

The resource type for the main segment will be set to match *type*. The resource ID number for the main segment is set to *id*. Using this option also sets the option `-xm c[oderesource]`.

The default is zero.

This option also turns on the `-xm c[oderesource]` option.

-safe filename

Mac OS

MWLinkPPC only.

When creating a code resource, tells the linker to use a safe code resource header. The file can be any file that contains a resource. The linker will look in *filename* to find a resource with the type and ID that match the type and ID of the current project, and use that resource to form a fat resource.

See also `-fat filename`, `-native`

-xm c[oderesource]

Mac OS

Generates a code resource without a code resource header. MWLinkPPC links object code into a PEF code resource. MWLink68K links object code into a 68K code resource.

The default creator for code resources is `'????'` and the default file type is `'APPL'`. If you are making a stand-alone code resource that you want to open in ResEdit, use `'RSED'` as the creator and `'rsrc'` as the file type.

See also `-native`, `-fat filename`, `-safe filename`

Code Fragment Control

-ad num

Mac OS

MWLink68K only.

This option controls the alignment of data for CFM68K. The value for *num* can be 1, 2, 4 or 8. The default is 4, which is what SOM expects.

-appSub[FolderID] id

Mac OS

When you create a shared library, under normal circumstances it can be in two places: in the Extensions folder or in the same folder as the application or other code that uses it. To tell the Code Fragment Manager to search elsewhere (perhaps in a folder created during an installation process), you can use this option.

This option stores the resource ID of an ‘alis’ resource (an alias) in your application's 'cfrg' resource, which describes the application’s

load directory. The default is zero, which specifies that there is no such directory.

This option has no effect unless `-cfrg on` is used.

-b

Mac OS

Expands uninitialized memory in the PEF file. By default, repetitive data in the Data Section is compressed to reduce file size. When loaded by the Code Fragment Manager, the Data Section is expanded to its full, run time size. The `-b` option expands the Data Section to its full size in the PEF file.

Use this option when creating all code resources.

-carr OSType

This option sets the `-codeorder` file creator. The default for OSType is 'MPS ' (use 'CWIE' for CodeWarrior type).

See also `-codeorder`.

-cfrg on/off

Mac OS

Enable or disable 'cfrg' creation. The default is "on." A 'cfrg' resource won't be made for code resources or static libraries.

-codeorder none/breadth[first]/depth[first]/segment/file=filename

Mac OS

MWLinkPPC only.

This option determines how routines are written to the PEF output file, with these possible values for the parameter:

- `none`—the routines are written in the same order as the input files (this is the default). You may also use `-codeorder source` to be the same as `-codeorder none`.
- `breadth[first]`—an example of a breadth-first ordering would be: A calls B & C, B calls D, then A, B, C, D. You may also use `-codeorder reference` to be the same as `breadth`.
- `depth[first]`—an example of a depth-first ordering would be: A calls B & C, B calls D, then A, B, D, C. You may also use `-codeorder staticall` to be the same as `-codeorder depthfirst`.
- `segment`—sorting is determined by `#pragma segment` directives in the source code. For information on `#pragma` segment, refer to the *C, C++, and Assembly Language Reference*..
- `file` *filename*—order routines as specified in the text file filename.arr. If filename.arr is missing, the linker will generate it.

This option is the same as `-sortcode`.

-cv num

Mac OS

Sets the current version number in your 'cfrg' resource to *num*. The default value is zero. The Code Fragment Manager uses this version number to check for code fragment compatibility at run time.

-dv num

Mac OS

Sets the definition number in the 'cfrg' resource for import libraries. The default value is zero. The linker uses this version number at link time when linking import libraries.

-e name

Mac OS

MWLinkPPC only.

Please see `-m[ain] name` option.

-export all/none

Mac OS and BeOS

Tells linker whether or not to mark all global variables and all routines in the PEF file table of contents as exportable. "All" makes all global variables and routines accessible to other external code fragments. "None" does not allow any global symbols or routines to be accessed from external code fragments.

The `-export all`, `-export none`, and `-@export filename.exp` are mutually exclusive. `-export none` implies `-export nopragma`. `-export all` implies `-export pragma`.

The default is `-export none`.

See also `-@export filename.exp`

-export [no]pragma

Mac OS and BeOS

The `-export pragma` option tells the linker to export symbols marked with `#pragma export on` and `#pragma export list` in source files. The default is `-export nopragma`. This option is independent of `-@export filename.exp`.

-export name [, name, …]

Mac OS and BeOS

Exports the specified global symbols, *name, name, …*. The specified global variables and routines are made accessible to external code fragments.

The default is

```
-export none
```

See also `-@export filename.exp`

-@export filename.exp

Mac OS

Marks the symbol names listed in file *filename.exp* for export.

The file *filename.exp* is a file containing a white-space separated list of symbols in the code fragment being created. Text following the '#' symbol on any line is ignored to the end of the line; it is treated as comments. The file name must end with the .exp extension.

These exported symbols will be accessible to external code fragments.

If *filename.exp* doesn't exist, the linker creates a file named *filename.exp* and stores a list of all symbols. You may pass more than one .exp file name to the linker.

See also `-export all/none`

-f filename

Mac OS and BeOS

MWLinkPPC and mwld only.

Same as `-@export filename.exp`.

BeOS

On the BeOS if `-f` is used more than onace all of the symbols from all of the files are exported.

-ib

Mac OS and BeOS

Before CodeWarrior 7, this option was `named -initbefore`. It specifies that the next library has the `initbefore` flag set to `TRUE`. It is the user's responsibility to ensure that the next file is a library. This option must physically come before the actual filename for which you want the `initbefore` flag set. This means that you cannot reliably put the option in `LinkOptions` if using the MPW development environment.

See also `-initbefore frag[,frag…]`.

-init name

Mac OS and BeOS

Specifies the name of the routine to be used as the code fragment's initialization routine.

See also `-e name`, `-m[ain] name`, `-term name`

BeOS

The default is `_init_routine_`

-initbefore frag[,frag…]

Mac OS and BeOS

Sets the `initbefore` flag for the shared library with fragment name *frag*. You can provide a comma-separated list of several fragments. This syntax changed in CodeWarrior 7. This option now precedes a comma-delimited list of fragment names. The option can occur before or after the shared library in the command line. `-initbefore` doesn't take the file name but instead takes the fragment name.

See also `-ib`, `-librename importname=libname.`

-librename importname=libname

Mac OS

MWLinkPPC only.

Change the name of an import library.

By default, MWLinkPPC strips the .xcoff from the end of an import library name to determine the fragment name. For example, InterfaceLib.xcoff becomes InterfaceLib. If the import library's name doesn't follow this convention, you must rename it yourself. `-librename` lets you rename a library on the command line. You can't be capricious in the fragment name used with `-librename`; the name must be the fragment name as found (case sensitive) in the 'cfrg' resource.

-m[ain] name

Mac OS and BeOS

Specifies *name* as the main entry point for the code being generated.

When linking classic (non-CFM) 68K object code with MWLink68K, CodeWarrior has a `__RuntimeModule__` that calls `main()`. If you have a function `main()`, you do not need to pass `-m`. If you do not have a function `main()`, use `-m` to identify your main entry point.

For PowerPC and 68K code fragments the default main entry point name is `__start`. If you are creating a PowerPC code resource in C, you will probably want to use `-m main`, if "main" is your main entry point.

For Code Fragment Manager applications, *name* is the main application entry point. The main application entry point is called after the code fragment's initialization routine.

For Code Fragment Manager shared libraries, the default main entry point is null, the equivalent of `-m ""`. The default is `__start` for applications and code resources, empty for a shared library and

drop in additions. The initialization and termination routines are called if they exist.

See also `-init name`, `-term name`

-mergefragment filename

Mac OS 68K and PowerPC

This option adds some features of the MPW Tool MergeFragment and is mostly useful when access to the MergeFragment tool is not possible.

When using this option, more than one code fragment can be put into the data fork of the output file, and the `'cfrg'` resource in the file will be modified accordingly. This option might be good for building FAT code fragments (code fragments that contain both 68K and PowerPC code).

An example of how to use this option to create a FAT code fragment at link time is shown here.

MWLink68K […options…] -xm s -o foo68K

MWLinkPPC […options…] -xm s -mergefragment foo68K -o foo

This example would link the 68K code fragment, storing it in the file foo68K. Then the PowerPC linker is invoked, which reads the code fragment from foo68K and writes it in with the PowerPC code fragment into the file foo, producing a FAT code fragment.

See also `-cfrg` (`-cfrg on` is required to use this option).

-model CFMflat[rf]/CFMflatdf

Mac OS

MWLink68K only.

Tells the linker to generate code to create a CFM68K binary and where to put the resulting code. For applications there are 2 possible

variations. The code can either go in the resource fork or the data fork. This option has two possibilities:

- `-model CFMflatdf`—This option causes the resulting binary file to be put in the data fork of the output file.
- `-model CFMflatrf or CFMflat`—the `CFMflat[rf]` option puts the code in the resource fork. The `CFMflat[rf]` option generates code that is put in the CODE 1 resource of the output file.

Any of these options are the same if creating a static or shared library. In other words, it makes no difference which option is specified when building a library, because the binary would always be written into the data fork.

-name PEFname

Mac OS

Sets the name of the PEF container in the 'cfrg' resource. If this option is not used or*PEFname* is empty, ("") then the 'cfrg' is given the linker's output name as the PEF container name.

-sharedata

Mac OS and BeOS

Sets the share option for the code fragment's data sections for global sharing. A single instance of the data sections are shared among all processes calling on the code fragment.

-sortcode

Mac OS and BeOS

MWLinkPPC and mwld only.

Sorts the code in the Code Section of the PEF file. Sorting is determined by `#pragma segment` directives in the source code. For information on `#pragma` segment, see the C/C++ Language Manual.

This option is the same as `-codeorder`.

-stacksize num

Mac OS

Specify the stack size in a 'cfrg' resource. The number is in kilobytes. For example, to have a 128K stack, pass `-stacksize 128`. The default is `kDefaultStackSize`. The default value is based on the current software and hardware configuration, typically 64K.

-term name

Mac OS and BeOS

Specifies the code fragment termination routine, *name*. The termination routine is called before a code fragment is unloaded by the Code Fragment Manager. Use the termination routine to release memory, resources, and any other objects allocated by the code fragment.

See also `-e name`, `-m[ain] name`, `-init name`

BeOS

The default name is _term_routine_.

-uv num

Mac OS

Set implementation version number to *num.* The code fragment's implementation number is checked at loading time to ensure the code fragment's caller and the code fragment are compatible. The default is zero.

See also `-weakimport`.

-weakimport

Mac OS and BeOS

Specifies that the next object file will have the `weakimport` flag set to `TRUE`. The user must ensure that the next object file is a library for which this flag works.

When a library is weak imported, it implies that the code referring to the library can work without it (either by disabling features or performing alternative actions). The code should check to ensure that the library is present at runtime. In the process, the Code Fragment Manager attempts to resolve the symbols in the weak imported library. If the Code Fragment Manager cannot resolve a weak imported symbol, it places an invalid address for the unresolved symbol in the code fragment's table of contents. This invalid address is defined as `kUnresolvedSymbolAddress` in the FragLoad.h Universal Header. Despite the fact that you cannot connect to the library, your code that refers to the library will launch and run.

In contrast, if a library is marked for strong import, if the Code Fragment Manager cannot resolve the symbols in that library it terminates the launch of your code. You cannot run, period.

If you use weak imports, it is your code's responsibility to check for unresolved symbols.

WARNING! Access to weak-imported objects will crash the computer.

-weaklib frag[,frag…]

Mac OS and BeOS

Set the weakimport flag for the shared library with fragment name *frag*. The list can be a comma separated list of fragments, and must be the shared library's name as found in the 'cfrg' resource.

See also `-librename importname=libname`, `-weakimport`.

-wi

Mac OS

Same as `-weakimport`.

Output File Control

-application

Mac OS

This is synonymous with `-xm a[pplication]`.

See also `-xm a[pplication]`, `-xm e[xecutable]`, `-codere-source`, `-sharedlibrary`.

BeOS

This is the default for BeOS.

-b

BeOS

This option causes the expansion of uninitialized storage.

-cv number

BeOS

Set the PEF (Program Executable Format) header current version number.

-c creator

Mac OS

Tells the linker to set the file creator to *creator*. The file creator must be a four character code. The default is `'????'`.

-fc filecreator

BeOS

This option specifies the output creator, with the default being '????'

-ft filetype

BeOS

This option specifies the output file type, with the default being 'BAPP'

-G

BeOS

This option causes the linker to produce a shared library, and is the same as `-sharedlibrary`.

-library

Mac OS

Synonymous with `-xm l[ibrary]`.

-libraryA4

Mac OS

MWLink68K only.

Synonymous with `-xm libraryA4`.

-lLibname

BeOS

Libname is the library to search. For example, if `-lpos` is passed on the command line the linker adds libpos.so to the link process.

-Llibrary

BeOS

This specifies the directory path to library files. Note that the forward slash "/" is the path delimiter for building pathnames.

-mpwtool

Mac OS

Synonymous with `-xm m[pwtool]`.

-o filename

Specifies the name of the output file as *filename*. You can provide a directory name, in which case the file will be named with the default file name (MWLink68K.out or MWLinkPPC.out), and appear in the specified directory.

-sharedlibrary

Mac OS and BeOS

MWLinkPPC and mwld only.

Synonymous with `-xm s[haredLibrary]`.

BeOS

Synonymous with -G.

-t type

Mac OS

Specifies the output file type. Type is a four character code. The default is `'APPL'`.

-xma

BeOS

Same as -xm a[pplication].

-xm a[pplication]

Mac OS and BeOS

Links object code into a double-clickable application. This is the default linker setting. The default type is 'APPL' and creator '????'. When you merge into an existing file, the original type and creator are used unless you specify new ones in the current project.

When creating a PowerPC application, be sure to add a `'cfrg'` resource to the resulting binary. This option is on by default. The MWLinkPPC tool will add this resource to the PowerPC application binary unless you tell it otherwise. Without a `'cfrg'` resource the Mac OS will not recognize the binary as a native PowerPC application.

See also `-application`, `-xm e[xecutable]`, `-coderesource`, `-sharedlibrary`,`-model CFMflat`.

-xm d[ropinaddition]

Builds a shared library as a drop in addition. The linker sets the flag in the 'cfrg' resource it creates for the library to indicate that it is a drop in addition. Refer to *Inside Macintosh: PowerPC System Software* for more information.

See also `-xm a[pplication]`, `-xm s[haredLibrary]`

-xm e[xecutable]

Mac OS

Synonymous with `-xm a[pplication]`.

See also `-application`, `-xm a[pplication]`, `-coderesource`, `-sharedlibrary`,`-model CFMflat`.

Listing 7.1 -xm gm restrictions

```
long myGlobal; /* Wrong */

void Foo( void )  /* Bad variable initialization */
{
char *greeting = "\pHi there!"; /* Wrong */
short i[3] = { 34, 45, 23 }; /* Wrong */

Bar( greeting, i );
}

void Foo( void ) /* Good variable initialization */
{
char *greeting; /* Do not initialize here. */
short i[3];

greeting = "\pHi there!"; /* OK */
i[0] = 34; i[1] = 45; i[2] = 23; /* OK */
Bar( greeting, i );
}
```

-xml

BeOS

Same as `-xm l[ibrary]`

-xm l[ibrary]

Mac OS and BeOS

Tells the linker to generate a CodeWarrior object code library. Type is always 'MPLF' and creator is always 'CWIE'. This is synonymous with `-library`. This is the normal option for building a library for either 68K or PowerPC.

If you wish to build a library that you will use in a 68K code resource, when you include CodeWarrior-created object files you don't need to do anything special, except that when you compile you must use `-a4`. However, if you wish to import an A5-based Apple OMF object file, you need to tell the linker that it is going into an A4-based library. You do so using the `-xm libraryA4` option.

When you use `-xm libraryA4`, the object file code will be converted on the fly so that all function calls use the A4 register. This conversion does not modify the use of A5 for global variables, so if the OMF code has global data, the conversion will not succeed and the code will crash.

-xm libraryA4

Mac OS

MWLink68K only.

See `-xm l[ibrary]`.

-xm m[pwtool]

Mac OS

Tells the linker to generate an MPW tool. Type is always `'MPST'` and creator is always `'MPS '`. Synonymous with `-mpwtool`.

-xms

BeOS

Same as `-xm s[haredlibrary]`

-xm s[haredLibrary]

Mac OS

Generates a Code Fragment Manager shared library (also called an *import library*). This is synonymous with `-sharedlibrary`. Unless you specify otherwise, type is 'shlb' and creator is 'cfmg'.

See also `-xm d[ropinaddition]`

Linker Operation Control

-fastlink on/off

Enables or disables faster linking. Faster linking uses more RAM instead of disk space. The `-fastlink off` option uses less RAM. The default is `-fastlink on`.

Using the `-fastlink on` option may force you to increase the memory requirements for the MPW Shell application from the Get Info window in the Finder.

Mac OS

The `-fastlink on` option is required if you are linking 68K object code with the MWLink68K tool and your object code uses function-level code segmentation based on `#pragma segment` directives in your source code. When creating segments based on function-level segmentation, the MWLink68K linker gathers individual functions in RAM, placing each one into the code segment specified by the `#pragma segment` directives. You can use `-fastlink off` if your 68K object code uses file-level segmentation.

See also `-s name`, `-sg newSegment = [oldSegment [,...] ]`.

-help

Display the usage information for the linker tool.

-link normal/fast[er]/slow[er]

Mac OS

The `-link normal` option needs lots of temporary memory but links in a small partition. This option is equivalent to `-fastlink off`.

The `-link faster` option needs lots of temporary memory plus requires a partition large enough to accomodate the entire SYM file being linked. This is about 10-20% faster than The `-link normal` linking. This option is equivalent to `-fastlink on`.

The `-link slower` option uses less temporary memory and links in a small partition, but takes about 2.5 times longer to link. It is useful for linking really large projects on machines with small amounts of memory. This option is available for MWLinkPPC only.

The default is `-link faster`.

BeOS

Only the `faster` and `normal` options are available for BeOS.

-maxerrors numbe

Limit the number of errors in the link process. The default is 100.

-[no]mf

Mac OS

Tells the linker whether to use MultiFinder memory (the Process Manager heap) when necessary. The default is `-mf`.

The `-nomf` option tells the linker to *never* use MultiFinder memory. The Shell's partition must hold all data.

-multidefs

Treat multiple definitions as warnings. The default is to treat multiple definitions as fatal errors.

-nodefaults

BeOS

Disables the defaults in the compiler and linker.

-noprejudge

Mac OS

MWLink68K only.

Conversions of MPW .o object files don't determine whether modules are code or data. Instead, the determination is based on usage in the program.

Diagnostic Message Control

-cmap type

Mac OS

This option sets the map file createor. The default is `'MPS '`, but you would want to use `'CWIE'` for a CodeWarrior file type.

See also `-map`.

-cexp type

Mac OS

This option sets the export file createor. The default is `'MPS '`, but you would want to use `'CWIE'` for a CodeWarrior file type.

-d

Mac OS

Prevents the generation of duplicate warnings.

See also `-w[arn].`

-map filename

Creates a link map file named *filename*. If you do not specify a file name, writes to the standard output.

A link map contains information about the symbols resolved at link time. As of CodeWarrior 7, the map information is now readable from within MPW and its format is much closer to the MPW map file format for code modules. You should find it easier to locate jump table and other data locations now. The file also correctly shows the resource IDs and names for code resources.

Mac OS

If you need to see the old IDE-style map, use `-mapIDE` *filename*.

See also `-cmap`.

BeOS

The recommended filename extension is .xMAP.

-mapIDE filename

MWLink68K only.

Creates a CodeWarrior IDE-style link map. See `-map filename`.

-msg option [,...]

MWLink68K and mwld only.

Set one or more linker message options.

Separate options with commas (`,`) to set more than one option. The message options are:

- `dup`—tells the linker to warn about duplicate symbols it finds in object code while linking. The linker may link the wrong symbol into the file it is creating.
- `nodup`—tells the linker not to warn about duplicate symbols
- `params`—tells the linker to warn about unsupported or substituted command-line options

- `noparams`—tells the linker to suppress linker warnings about unsupported or substituted command-line options
- `warn`—tells the linker to issue linker warnings
- `nowarn`—tells the linker to suppress linker warnings

-nodup

BeOS

Prevent duplicate warnings.

-p

Shows progress information as the linker processes files.

-version

Display the version information for the linker tool.

-w[arn]

Suppresses warning messages while linking. This is on by default, so that warning messages are displayed.

Debugging Control

-csym ‘creator’

Mac OS

Tells the linker to set the creator type of the SYM debugging file. Default value is for the CodeWarrior debugger.

-g

BeOS

Generate symbolic debugging information, equivalent to `-sym full`.

-osym filename

Mac OS

Saves debugging symbolic file as *filename*: The default filename is `filename`.xSYM for PowerPC and *filename*.SYM for 68K.

See also `-o filename`

-sym on/off/full/fullpath/tdb/3.2/3.3

The `-sym on` option tells the linker to create a SYM file. The debugging information is stored in a .SYM (for 68K) or .xSYM (for PowerPC) file.

The `-sym fullpath` option differs from the `-sym on` option in that it stores the full path name for every source file in the SYM file.

The `-sym 3.2` and `-sym 3.3` options specify the version of SYM information to generate. These options are available for MWLink68K only.

Note that -sym 3.3 is the default.

Mac OS

Under the Mac OS, the `-sym full` and `-sym tdb` options are identical. They do all of the above, and (for 68K code) tells the linker to link runtime code that supports Jasik's The Debugger. For PowerPC code, `-sym full` and `-sym tdb` are identical to `-sym full-path`.

See also `-osym filename.`

-tdb

Mac OS

MWLink68K only.

Tells the linker to link runtime code that supports Jasik's The Debugger. This option does not generate SYM information.

See also `-sym on/off/full/fullpath/tdb/3.2/3.3`.

8

PowerPC Assembler

This chapter describes the PPCAsm PowerPC assembler and its options. This tool turns PowerPC assembly language into object code.

Overview of the PowerPC Assembler

The assembler for MPW—PPCAsm—translates PowerPC assembly language into object code. This tools perform functions analogous to the asm keyword in the CodeWarrior 68K compiler syntax. PPCAsm is part of Apple Computer's suite of command line tools.

The correct syntax for the assembler is

PPCAsm [options…] filename[…] > listing ≥ progress

The tool generates object code for the specified file using the specified options. The results are sent to the standard output file. This tool is straightforward, with very few options.

This chapter shows the options available for use on the command line for MPW or ToolServer. For more complete documentation on the assembler and syntax issues refer to *PowerPC Assembler Notes* in the *C Compiler Guide* or the Apple MPW documentation on the CodeWarrior CD.

Assembler options

-auxheader modtype

Generate an auxiliary header (XCOFF) with a module having type *modtype*.

modtype = 1R (single use), RE (reusable), or RO (read only)

The default is to not generate this header.

-blocksize size

Specify the input and output buffer size in bytes. The default size is 512 bytes.

-check

Check for syntactical errors only and suppress the generation of an output object file.

-d[efine] name[=value]

Assign a decimal integer *value* to *name.* This is equivalent to placing the assembly directive *name* SET *value.* at the beginning of the file. More than one name may be defined by specifying multiple -d options or multiple *name* assignments separated by commas.

The default value for a *name* is 1.

-depth value

Change the maximum depth of the macro call stack to *value*. The default for *value* is 64.

-dialect type

Specifies whether to expect POWER or PowerPC assembly instruction mnemonics.

values = Power (POWER instructions) or PowerPC (PowerPC instructions)

The default for this option is PowerPC.

-i pathname

This option specifies a search *pathname* for include files. If the include files are not found along the specified *pathname* the following directories are searched if a partial *pathname* is specified (no colons in the *pathname* or a leading colon are allowed):

- the directory containing the current source file
- other directories specified in the -i option as listed
- the directories specified in the MPW or ToolServer shell variable {PPCAsmIncludes}

-l

This option writes the assembly listing to standard output unless -lo is specified.

-lo listingName

This option writes the assembly listing to *listingName*.

-o objectName

This option writes the object file to *objectName*.

-p

This option writes progress, version and summary information to the diagnostic output.

-t

This option displays only the time and number of lines to the diagnostic output. This option works regardless of whether -p is specified.

-w

Suppress warning messages.

9

Metrowerks Disassemblers

This chapter describes the Metrowerks disassemblers and their options.

Overview of Metrowerks Disassemblers

The Metrowerks disassemblers for MPW—MWDumpPPC and MWDump68K—translate PowerPC or 68K object code into assembly language. These tools perform functions analogous to the Disassemble command in the CodeWarrior IDE's Project menu.

The correct syntax for the disassemblers is

MWDumpPPC [options…] [file]

MWDump68K [options…] [file]

Each tool generates a disassembly for the specified file using the specified options. The results are sent to the standard output file. These tools are straightforward, with very few options.

Disassembler options

-a4

For MWDump68K only.

This option tells the disassembler to convert an A5-based Apple OMF file to be A4-based for use in code resources. Pass `-a4` to see how the linker is going to interpret the file in an A4-based world.

The default is to leave the object code uninterpreted.

The intricacies of this issue are fully discussed at "-xm l[ibrary]" on page 35.

-d

Don't dump object code.

-fmt fmtparams

For MWDumpPPC only.

Same as `-format [no]x`.

-format [no]x

For MWDumpPPC only.

This option allows you to turn off the use of extended mnemonics in the PowerPC disassembly.

The default is `-format x`.

-h

This option tells the disassembler to output code modules without offset or hexadecimal values.

The default is to output offset and hexadecimal values.

Use this option to copy and paste the disassembler's output to use as assembler source code. Some parts of this output will have to be tweaked to be used with CodeWarrior. Refer to the *C, C++, and Assembly Language Manual* for more information on assembly language programming using CodeWarrior.

-help

Emits usage information for the disassembler.

-m name

Tells the dissassembler to dump only module or entry point 'name';

- Classic 68K—just use symbol *name*
- CFM68K—use _$*name*
- PowerPC—add a dot (`.`) to the beginning of *name*

The default is to output all modules in a file.

-mix

This option allows you to see source and assembly simultaneously in the same dump file, under certain circumstances.

Because the disassemblers read the sym information stored in the object file to find the source offsets, you must compile with `-sym full`. You should use `-sym full` so that the disassembler knows where to find the file. Having optimizations off is not required and is often not necessary to follow the code.

-n

Tells the disassembler to output symbolic names only.

-noheader

Suppresses the source file name in the output.

-p

Display progress information as the disassembly proceeds.

-sym on/off/full

Controls the output of symbolic debugging information.

`-sym on` and `-sym full` tell the disassembler to output symbolic debugging information.

`The -sym off` option, the default, tells the disassembler not to output symbolic debugging information.

-version

This option displays the dissasembler's version information.

-xtables on/off

Enable or disable showing exception tables.

The default is `-xtables on`.

10

MPW Tools Compared

This chapter contains a series of tables detailing on an option- by-option basis how the CodeWarrior MPW tools compare functionally to other compilers and linkers.

Overview of MPW Tools Compared

The tools compared are:

- C and MWC68K compilers
- CPlus and MWC68K compilers
- SC/Cpp and MWC68K compilers
- PPCC and MWCPPC compilers
- MrC/MrCpp and MWCPPC compilers
- Link and MWLink68K linkers
- PPCLink and MWLinkPPC

If you are familiar with one or another of these other tools, the associated tables should help you understand how the CodeWarrior tools operate. In most cases, the functionality is identical. In some cases, the CodeWarrior tools operate differently.

The CodeWarrior MPW tools have additional options not found in the foreign tools. For full information on the options available for the CodeWarrior MPW compilers and linkers, see the other chapters of this manual.

This chapter has a section devoted to each tool comparison. Each section begins with a brief description of the principal differences

between the two tools. Following that description is a table with a comprehensive listing of each of the foreign tool's options.

Each comparison table has four columns.

The first column lists the foreign tool's options.

The second column answers the question, "What would happen if I passed this option to the CodeWarrior tool?" There are three possible answers:

- same—the same behavior occurs in CodeWarrior's tool
- ignored—CodeWarrior's tool ignores this option (you will usually see a warning in the standard output file)
- a comment—details explaining subtle differences between the behavior of the option in the two tools

The third column answers the question, "What is the preferred CodeWarrior option to get this same feature?" You are likely to see one of three answers:

- same—the CodeWarrior tool has the same option
- none—there is no comparable option in the CodeWarrior tool
- option—the preferred option for the CodeWarrior tool

For full details on the CodeWarrior option suggested, refer to the other chapters in this manual that detail the compiler and linker options.

The fourth and last column of the comparison table contains a description of the foreign tool's native behavior. This is derived from that tool's help file.

C and MWC68K compilers

There are several important differences between C and MWC68K. The differences are shown in Table 10.1.

Be aware that:

- The `-b3` option in MWC68K does nothing about providing A5-relative references for function addresses.
- The C option `-bigseg` is not implemented in MWC68K; use `-model farCode` instead.
- The C option `-m` is quite different; MWC68K uses `-model farData`, `-model farStrings`, `-model farVTables.`
- MWC68K ignores all of the C `-trace` options.
- The C option `-elems881` is similar to the MWC68K option `-mc68881`.
- The C option `-mbg <n>` is the same as the MWC68K option `-mbg on`.
- MWC68K ignores `-k directory` and `-y directory`.
- The `-e` and `-e2` options behave somewhat differently.
- MWC68K supports `-mf`. It is the default value.
- The `-sym on|full` options are quite different. See Table 1.1 for details.

Table 10.1 Comparing C and MWC68K

C options	Behavior in MWC68K	MWC68K preferred	C's definition
-b	same	same	put string constants into code and generate PC-relative references for function addresses
-b2	same	same	implies '-b' above, and allows string constants to be overlaid
-b3	same as -b2	-b2	overlaid string constants in code (but A5-relative references for function addresses)
-bigseg	ignored	-model farCode	generate single large (>32K) code segment, 68020 only
-c	same	same	syntax check only, don't create object file

C options	Behavior in MWC68K	MWC68K preferred	C's definition
-d name	same	same	equivalent to: #define name 1
-d name=string	same	same	equivalent to: #define name string
-e	same as -e2	-e	write preprocessor results to output
-e2	strips comments and blank lines	same	implies '-e' above, and strips comments lines
-elems881	same as -mc68881	-mc68881	generate MC68881 code for transcendentals
-i directory	same	same	search for includes in directory
-k directory	ignored	none	create load/dump files in directory
-m	ignored	-model farData	generate 32-bit references for data (less efficient code) -model strings, and -model farVTables
-mbg ch8	same	same	v2.0 compatible MacsBug symbols
-mbg off	same	same	no MacsBug symbols in the code
-mbg on | full	same	same	full MacsBug symbols
-mbg <n>	same as on	-mbg full	MacsBug symbols to length <n> (<n> can be 0…255)
-mc68020	same	same	generate MC68020 code
-mc68881	same	same	generate MC68881 code for arithmetic operations

C options	Behavior in MWC68K	MWC68K preferred	C's definition
-model farCode | farData | far	same but CodeWarrior also makes distinction between near and far strings and VTables	same	generate load-time relocatable 32-bit references for code, data, or both
-model nearCode | nearData | near	same but CodeWarrior also makes distinction between near and far strings and VTables	same	generate 16-bit references for code, data, or both (the default)
-n	same as `-mpw_chars` and -relax_pointers CodeWarrior is not as forgiving as C	-relax_pointers	turn pointer assignment incompatibility errors into warnings
-notonce	same	same	do not automatically suppress multiple inclusion of files
-o objname	same	same	generate code in file or directory objname
-opt off	same	same	don't apply code optimizations
-opt on | full	same	same	choose level of code optimization (on is default); can modify with [,nopeep] [,nocse] (no peephole, no common subexpression)

C options	Behavior in MWC68K	MWC68K preferred	C's definition
-p	same but CodeWarrior is not as verbose	-p	write progress information to diagnostic
-r	same	same	warn on calling a function that has no definition
-s segment	same	same	generate code in segment
-sym off	same	same	don't generate SADE records
-sym on | full	same, but nolines, notypes,novars are ignored	-sym full	generate SADE records; can modify with [,nolines] [,notypes] [,novars]
-t	same	same	write compilation time to diagnostic
-trace always | never	ignored	none	enable/disable calls to profiler or MacApp debugger (overrides pragma trace on | off)
-trace on | off	ignored	none	enable/disable calls to profiler or MacApp debugger (subject to pragma trace on | off)
-u name	same	same	equivalent to: undef name
-w	same	-w off	suppress warnings
-w2	same	-w full	set warning level
-warnings on | off	same	same	set warning level; “on” is the default
-warnings full	same	same	set warning level
-y directory	ignored	none	create temporary files in directory

CPlus and MWC68K compilers

There are several important differences between CPlus and MWC68K. The differences are shown in Table 10.2.

Be aware that:

- The `-b3` option in MWC68K does nothing about providing A5-relative references for function addresses.
- The C option `-bigseg` is not implemented in MWC68K; use `-model farCode` instead.
- The C option `-m` is quite different; MWC68K uses `-model farData`, `-model farStrings`, `-model farVTables.`
- MWC68K ignores all of the C `-trace` options.
- The C option `-elems881` is same as the MWC68K option `-mc68881`.
- The C option `-mbg <n>` is the same as the MWC68K option `-mbg on`.
- MWC68K ignores `-k directory` and `-y directory`.
- The `-e` and `-e2` options behave somewhat differently.
- MWC68K supports `-mf`. It is the default value.
- The `-sym on|full` options are quite different. See Table 10.2 for details.

Table 10.2 Comparing CPlus and MWC68K

CPlus options	Behavior in MWC68K	MWC68K preferred	CPlus's definition
-a	same	-ansi off	force ANSI-style intermediate C code (default)
-a0	same	-ansi off	force K&R-style (pre-ANSI) intermediate C code
-a1	same	-ansi off	force ANSI-style intermediate C code (default)

CPlus options	Behavior in MWC68K	MWC68K preferred	CPlus's definition
-b	same	same	passed through to C compiler (put string constants into code and generate PC-relative references)
-b2	same	same	passed through to C compiler (implies '-b' above, and allows string constants to be overlaid)
-b3	same as -b2	-b2	passed through to C compiler (overlaid string constants in code—but not PC-relative references.)
-c	same	same	intermediate code to standard output; no object file
-comp name options	ignored	none	use a different C compiler in "name" and give it "options"
-d name	same	same	equivalent to: #define name 1
-d name=string	same	same	equivalent to: #define name string
-dump filename	same	-precompile filename	save state of C++ compilation in filename
-dumpc filename	same not compressed	-precompile filename	save state of C++ compilation in filename (compressed)
-e	same as -e2	same	write preprocessor results to output
-e2	strips comments and blank lines	same	implies '-e' above, and strips comments
-elems881	turns on -mc68881	-mc68881	generate MC68881 code for transcendentals

CPlus options	Behavior in MWC68K	MWC68K preferred	CPlus's definition
-f filename	ignored	none	act as if input comes from named file when it actually is from stdin
-f1	ignored	none	send text version of C code to C compiler (no token stream)
-f2	ignored	none	send token stream of C code to C compiler (default)
-i directory	same	same	search for includes in directory
-k directory	ignored	none	create load/dump files in directory
-l	ignored	none	in C output generate line nn directives as nn
-l0	ignored	none	in C output generate no line nn directives at all
-load filename	same	-prefix filename	load saved C++ compilation state from filename
-m	ignored	-model farData, -model strings, and -model farVTables	passed to C compiler: generate 32-bit references for data
-mark fcts | types | data | globals	ignored	none	create MPW markers for the specified items in file being compiled
-mark all	ignored	none	create MPW markers for all of the above
-maxerrors <n>	same	same	specify that CFront abort after reporting <n> errors (default 12)
-mbg ch8	same	same	v2.0 compatible MacsBug symbols

CPlus options	Behavior in MWC68K	MWC68K preferred	CPlus's definition
-mbg off	same	same	no MacsBug symbols in the code (link/unlk only if necessary)
-mbg on | full	same	same	full MacsBug symbols
-mbg <n>	same as on	-mbg full	MacsBug symbols to length <n> (<n> can be 0…255)
-mc68020	same	same	generate MC68020 code
-mc68881	same	same	generate MC68881 code for arithmetic operations
-mf	same; default	same	use MultiFinder memory if there is not enough MPW memory
-model near | far	same	same	use 32-bit addresses for global code and data (model far) or A5 references (model near) -- "near" is default
-mtbl0	ignored	none	suppress output of method tables for Object Pascal classes
-mtbl1	ignored	none	force output of method tables for Object Pascal classes
-n	same as -mpw_chars or -relax_pointers CodeWarrior is not as forgiving	-relax_pointers	passed to C compiler: turn pointer assignment incompatibility errors into warnings
-o objname	same	same	generate code in file or directory objname
-opt off	same	same	passed to C compiler: don't apply code optimizations

CPlus options	Behavior in MWC68K	MWC68K preferred	CPlus's definition
-opt on | full	same	same	passed to C compiler: choose level of code optimization (on is default);
-p	same but CodeWarrior is not as verbose	same	write progress information to diagnostic
-patch	ignored	none	use AT&T's "patch" mechanism for static constructors (use with -comp)
-s segment	same	same	generate code in segment
-sym off	same	same	no SADE records
-sym on | full	same; however nolines, notypes,novars are ignored	-sym full	full SADE records; can modify with [,nolines] [,notypes] [,novars]
-t	same	same	passed to C compiler: write compilation time to diagnostic
-trace on	ignored	none	passed to C compiler: generate tracing code—calls to %_EP and %_BP
-trace off	ignored	none	passed to C compiler: generate no tracing code (default)
-trace always	ignored	none	passed to C compiler: always generate tracing code—calls to %_EP and %_BP
-trace never	ignored	none	passed to C compiler: never generate tracing code (default)
-u name	same	same	equivalent to: undef name
-vtbl0	ignored	none	suppress output of virtual tables for ordinary classes

CPlus options	Behavior in MWC68K	MWC68K preferred	CPlus's definition
-vtbl1	ignored	none	force output of virtual tables for ordinary classes
-w	same	-w off	suppress warnings
-w1	same as -w on	-w on	generate additional warnings from CFront
-w2	same as -w full	-w full	generate maximum warnings from CFront and the C compiler
-w3	same as -w on, nounused-var,nounuse-darg	-w full	passed through to C compiler (suppresses "unused" warnings)
-warnings on | off | full	same	same	passed to C compiler: set warning level; "on" is the default
-x filename	ignored	none	generate C code for a different processor, using size and alignment info in filename
-y directory	ignored	none	create temporary files in directory
-z0	same as -inline off	-inline off	force inline functions to be non-inline
-z3	ignored	none	suppress name encoding of local vars and struct members (default)
-z4	ignored	none	encode names of local vars and struct members
-z5	ignored	none	don't strip unused static objects that have constructors or destructors
-z6	same as -enum int	-enum int	force enumerated values to always be int variables

CPlus options	Behavior in MWC68K	MWC68K preferred	CPlus's definition
-z7	ignored	none	relax requirement on static class member initialization
-z15	ignored	none	optimize nested inline constructor-allocator code
-z17	same as -inline on but no guarantees	-inline on	allow inline expansion of functions of any size

SC/Cpp and MWC68K compilers

There are some important differences between SC/Cpp and MWC68K. The differences are shown in Table 10.3.

- MWC68K doesn't number warnings. You can enable or disable certain types of warnings by passing a modifier to -w on or -w full.
- The CW8 version of MWC68K supports the flat model of CFM68K so there are some options that aren't available.
- MWC68K ignores the `-trace` options.

Table 10.3 Comparing SC/Cpp and MWC68K

SC/Cpp options	Behavior in MWC68K	MWC68K preferred	SC/Cpp's definition
-align power	same	same	set alignment within structures; mac68k is default
-align mac68k	same	same	
-align byte	ignored	none	
-ansi off | on | relaxed | strict	same	same	ANSI compatibility; off is default

SC/Cpp options	Behavior in MWC68K	MWC68K preferred	SC/Cpp's definition
-auto_import size	ignored	none (cfm)	if cfm model, flag data of 'size' bytes as imported
-b	same	same	produce pc-relative strings and functions
-b2	same	same	produce pc-relative strings and functions, overlay equal strings
-b3	same as -b2	none	produce pc-relative strings
-b4	same		default
-bigseg	ignored	none (cfm)	generate code for single large segment
-char signed	ignored; default	none	default sign of chars
-char unsigned	ignored	none	
-char unsignedX	ignored	none	
-d name	same	same	equivalent to: #define name 1
-d name=string	same	same	equivalent to: #define name string
-D name	same	same	equivalent to: #define name 1
-D name=string	same	same	equivalent to: #define name string
-dump[c] filename	same	-precompile filename	save state of C++ compilation in filename
-e	same	same	write preprocessor results to output
-elems881	turns on -mc68881	-mc68881	generate MC68881 code for transcendentals (for 68K only)

SC/Cpp options	Behavior in MWC68K	MWC68K preferred	SC/Cpp's definition
-enum min | int	same	same	set enum size
-fatext	same	same	Use .68k.o and/or .ppc.o extensions instead of .o
-frames	same (68K only)	-a6	always generate stack frames
-i[path][,path]	same	same	search for header files along path
-inline on | all | none | off	same	same	expand inline functions
-j0 | 1 | 2	ignored	none	recognize 2 byte Japanese | Chinese, Taiwanese | Korean characters
-l filename	ignored	none	generate a list file
-load[c] file-name	same	-prefix file-name	load saved C++ compilation state from filename
-mbg on | full | off	same	same	on=full generate MacsBug symbols (default)
-mc68020	same	same	generate MC68020 code (for 68K only)
-mc68030	same as -mc68020	-mc68020	generate MC68030 code (for 68K only)
-mc68040	same as -mc68020	-mc68020	generate MC68040 code (for 68K only)
-mc68881	same	same	generate MC68881 code for arithmetic operations (for 68K only)
-model far-Code	same	same	generate load-time relocatable 32-bit references for code (for 68K only)

SC/Cpp options	Behavior in MWC68K	MWC68K preferred	SC/Cpp's definition
-model far-Data	same	same	generate load-time relocatable 32-bit references for data (for 68K only)
-model far	same but includes far strings and VTables	same	generate load-time relocatable 32-bit references for data and code (for 68K only)
-model nearCode	same	same	generate 16-bit references for code
-model near-Data	same	same	generate 16-bit references for data
-model near	same but includes near strings and VTables	same	generate 16-bit references for code and data (default)
-model cfmseg	same as -model cfm-flat	-model cfm-flat	request generation of multi-segment NuRuntime model supporting shared libraries
-model cfmflat	same	same	request generation of single segment NuRuntime model supporting shared libraries
-nomfmem	same	-nomf	inhibit use of MultiFinder memory
-notonce	same	same	allow multiple copies of header files
-o file-name | dirname:	same	same	set output file or directory
-onefrag	ignored	none	assume any C++ constructs for Nu-Runtime model are in the same fragment
-opt all	same	same	use all optimizations except space

SC/Cpp options	Behavior in MWC68K	MWC68K preferred	SC/Cpp's definition
-opt off	same	same	perform minimal optimizations
-opt none	same	off	perform minimal optimizations
-opt space	same	size	favor space optimizations over time optimizations
-opt time	same	speed	favor execution time optimizations over space optimizations
-opt speed	same	same	favor execution time optimizations over space optimizations
-p	same	same	print progress messages
-paslinkage	ignored	none	Pascal linkage option
-s[eg] segname	same	-s segname	assign code to segment (for 68K only)
-sym off	same	same	no symbolic records
-sym on	same	same	create symbolic records
-sym full	same but adds full paths	same	full symbolic records
-sym full[,notypes]	same as -sym on	none	omit type information
-sym full[,nolines]	same as -sym on	none	omit source line information
-sym full[,novars]	same as -sym on	none	omit variable information
-sym full[,altypes]	same as -sym on	none	include ALL type information
-trace on	ignored	none	generate tracing code—calls to %_EP and %_BP

SC/Cpp options	Behavior in MWC68K	MWC68K preferred	SC/Cpp's definition
-trace off	ignored	none	generate no tracing code (default)
-trace always	ignored	none	always generate tracing code—calls to %_EP and %_BP
-trace never	ignored	none	never generate tracing code (default)
-u	ignored	-u macro	suppress predefined non-ANSI macros
-w on	same	same	print all warnings (default)
-w off	same	same	don't print warnings
-w [is]err[or]	same	-w iserror	warnings are treated as errors
-w [,...]	ignored	-w on	don't print warning N
-x	same	same	continue past error limit
-xa templatespec	ignored	none	instantiate a template with arguments
-xi p[ublic]|s[tatic]|e[xtern]	ignored	none	instantiate templates in this file accordingly

PPCC and MWCPPC compilers

There are some important differences between PPCC and MWCPPC.

- MWCPPC supports `-mf`. It is the default value.
- The PPCC options `-export_list filename` and `-shared_lib_export on|off` are ignored.
- Using MWCPPC, exported symbols are stored in the .o file to be used by the linker if desired.
- Sym generation in MWCPPC does not turn off optimizations.

Optimization Differences

Optimizations are handled differently in MWCPPC. With PPCC, the highest level of optimization is named `-opt speed`. This is approximately equal to `-opt all` in MWCPPC. The PPCC option `-opt local` is approximately equivalent to CodeWarrior's `-opt on,l2`.

NOTE: In CodeWarrior, the speed optimization is really a guideline to the compiler for which direction to go when performing other optimizations. Given an alternative between faster or smaller code, the “speed” option guides the compiler on which route to take. This is very different from PPCC’s use of the term “speed,” which encompasses all available optimizations on that compiler.

Because PPCC optimizing options `nounroll` and `norep` actually remove PPCC's speed optimizations, using them with CodeWarrior would be confusing. In CodeWarrior these changes would actually add optimizations to speed, not remove them.

Table 10.5 compares the tools option-by-option.

In MWCPPC:

- `-opt on` (the default setting) is `-opt speed,peep,global, schedule,l1`
- `-opt all` is `-opt speed,peep,global,schedule,l4`
- `-opt off` turns all optimizations off
- options like `peep`, `global`, `schedule`, `speed`, and `size` affect only that specific type of optimization

Table 10.4 Comparing Optimizations

PPCC	MWCPPC
nothing	-opt speed,peep,l1
-opt off	-opt speed,l1

PPCC	MWCPPC
-opt local	-opt speed,peep,global,schedule,l2 (same as -opt on,l2)
-opt speed	-opt speed,peep,l1
-opt size	-opt speed,peep,l1
-opt [speed,] nounroll	-opt speed,peep,global,schedule,l2 (same as -opt on,l2)
-opt [speed,] norep	-opt speed,peep,global,schedule,l3 (same as -opt on,l3)

Table 10.5 Comparing PPCC and MWCPPC

PPCC options	Behavior in MWCPPC	MWCPPC preferred	PPCC's definition
-align power	same	same	specify power alignment for data structures (default)
-align mac68k	same; default	same	specify 68K alignment for data structures
-appleext on	same	same	enable Apple's language extensions.
-appleext off	same	same	disable Apple's language extensions (default)
-char signed	ignored; default	none, default	treat char type as signed. (default)
-char unsigned	ignored	none	treat char type as unsigned
-d name	same	same	define a preprocessing symbol. Equivalent to #define name 1.
-d name=value	same	same	define a preprocessing symbol. Equivalent to #define name value.
-dialect ansic	same	-dialect c	set source language dialect.

PPCC options	Behavior in MWCPPC	MWCPPC preferred	PPCC's definition
-dialect cplus	same	same	Files ending in ".cp" use "cplus" dialect by default.
-dialect krc	same as -dialect c but adds -proto ignoreold-style	-dialect c	All others are have the default "ansic". See also: -strict.
-e	same as -e2	same	write preprocessor results to standard output
-e2	strips comments and blank lines	same	implies -e and also strips comments.
-enum int	same	same	use int-sized enumeration types.
-enum min	same	same	use minimum-sized enumeration types, same as 68K (default)
-errsource on | off	ignored	none	do | don't include offending source line in error messages. (default: on)
-export_list filename	ignored	none	specify a file in which to list exported names. For use with -shared_lib_export. (default: <source filename>.x)
-fp_contract on | off	same	same	on: may generate Multiply Add Fused instructions (default: on)
-i directory	same	same	search for include files in directory.
-inline none	same	same	disable all inlining.
-inline all	same	same	enable all inlining.
-inline cplus	same as on	-inline on	enable C++ inlining

PPCC options	Behavior in MWCPPC	MWCPPC preferred	PPCC's definition
-inline noc-plus	same as off	-inline off	disable C++ inlining
-inline user	same as on	-inline on	enable implicit inlining
-inline nouser	same as off	-inline off	disable implicit inlining
-inline lib	same as on	-inline on	enable lib function inlining
-inline nolib	same as off	-inline off	disable lib function inlining
-ldsize 64 | 128	only support 64; warning for 128	same	select long double size (default: 128-bit)
-list pre	ignored	none	send listing to stdout with preprocessing (default: no listing)
-list nopre	ignored	none	as above, but without preprocessing
-maxerrors n	same	same	set max error limit to n. (default: 20)
-nestcom	ignored	none	allow nested comments (default: no nested comments)
-nocodegen	same	-c	perform syntax and semantic checking only; don't generate code. (default: generate code)
-o pathname	same	same	send generated code to pathname. (default: <source filename>.o)
-opt off	same -sym doesn't matter	same	suppress all optimizations. (default with -sym on)
-opt local	same -sym doesn't matter	-opt on	perform local optimizations (default with -sym off)

PPCC options	Behavior in MWCPPC	MWCPPC preferred	PPCC's definition
-opt size	same -sym doesn't matter	same	optimize for size.
-opt speed	see comments above table	-opt full	optimize for speed.
-p	same	same	write progress information to stdout
-shared_lib_export on	ignored	none	enable code generation for exporting names from a shared library.
-shared_lib_export off	ignored	none	disable code generation for exporting names from a shared library.
-strict on	same	same	enable/disable strict enforcement of the source language dialect semantics.
-strict off	same	same	(default) see also: -dialect.
-sym on	same but doesn't disable optimizations	same	enable generation of symbolic debugging information.
-sym on, alltypes	same; alltypes ignored	-sym full	generate debugging info for all types (even if unreferenced)
-sym full	same	same	same as -sym on.
-sym off	same	same	disable generation of symbolic debugging information. (default)
-u name	same	same	undefine a preprocessor symbol, equivalent to `#undef name`

PPCC options	Behavior in MWCPPC	MWCPPC preferred	PPCC's definition
-w off	same	same	suppress all warnings.
-w conformance	same	-w on	generate dialect conformance warnings
-w usage	same	-w on	generate "questionable usage" warnings (default)
-w all	same	same	generate both usage and conformance warnings
-y directory	ignored	none	write compiler temporary files to directory (default:{tempfolder})

MrC/MrCpp and MWCPPC compilers

Important differences between MrC/MrCpp and MWCPPC

- MWCPPC supports `-mf`. It is the default value.
- Using MWCPPC, exported symbols are stored in the .o file to be used by the linker if desired.
- MWCPPC doesn't number warnings. You can enable or disable certain types of warnings by passing a modifier to -w on or -w full.
- Sym generation in MWCPPC does not turn off optimizations.

Optimization Differences

Optimizations are handled differently in MWCPPC. With MrC/MrCpp, the highest level of optimization is named `-opt speed`. This is approximately equal to `-opt all` in MWCPPC. The MrC/MrCpp option `-opt local` is approximately equivalent to CodeWarrior's `-opt on,l2`.

NOTE: In CodeWarrior, the speed optimization is really a guideline to the compiler about the direction in which to go when performing other optimizations. Given an alternative between faster or smaller code, the “speed” option guides the compiler on which route to take. This is very different from MrC/MrCpp’s use of the term “speed,” which encompasses all available optimizations on that compiler.

Because MrC/MrCpp optimizing options `nounroll` and `norep` actually remove MrC/MrCpp's speed optimizations, using them with CodeWarrior would be confusing. In CodeWarrior these changes would actually add optimizations to speed, not remove them.

In MWCPPC:

- `-opt on` (the default setting) is `-opt speed,peep,global, schedule,l1`
- `-opt all` is `-opt speed,peep,global,schedule,l4`
- `-opt off` turns all optimizations off
- options like `peep`, `global`, `schedule`, `speed`, and `size` affect only that specific type of optimization

Table 10.7 compares the tools option-by-option.

Table 10.6 Comparing Optimizations

MrC/MrCpp	MWCPPC
nothing	-opt speed,peep,l1
-opt off	-opt speed,l1
-opt local	-opt speed,peep,global,schedule,l2 (same as -opt on,l2)
-opt speed	-opt speed,peep,l1
-opt size	-opt speed,peep,l1
-opt [speed,] nounroll	-opt speed,peep,global,schedule,l2 (same as -opt on,l2)
-opt [speed,] norep	-opt speed,peep,global,schedule,l3 (same as -opt on,l3)

Table 10.7 Comparing MrC/MrCpp and MWCPPC

MrC/MrCpp options	Behavior in MWCPPC	MWCPPC preferred	MrCpp's definition
-align power	same	same	specify power alignment for data structures (default)
-align mac68k	same	same	specify 68K alignment for data structures
-ansi off	same	same	don't enforce strict ANSI compatibility (see the Symantec C/C++ for MPW Users Guide.)
-ansi on | relaxed	same	same	size of enumerated values is determined by minimum size needed to store the enum values. (See MrC and MrCpp chapter in E.T.O./MPW Pro Release Notes for complete list of features and restrictions.)
-ansi strict	same	same	same as -ansi on, but size of enumerated values is always an int

MrC/MrCpp options	Behavior in MWCPPC	MWCPPC preferred	MrCpp's definition
-char signed	ignored; default	none	treat char type as signed (default)
-char unsigned	ignored	none	treat char type as unsigned
-char unsignedx	ignored	none	treat char type as unsigned, and sign extend the values
-c	same	same	Syntax check only. No XCOFF file is generated.
-currdir	ignored	none	use current (working) directory for the compiler temporary and output files. The -y option has precedence over the -currdir option.
-d[efine] name	same	same	define a preprocessing symbol. Equivalent to #define name 1.
-d[efine] name=value	same	same	define a preprocessing symbol. Equivalent to #define name value.
-dump[c] filename	same	-precompile filename	save the state of the compilation in the specified file
-e	same; ignores -l and removes blank lines and comments	same	write preprocessor results to stdout (-l must also be specified)
-enum int	same	same	use int-sized enumeration types
-enum min	same	same	use minimum-sized enumeration types, same as 68K (default unless -ansi strict is on)

MrC/MrCpp options	Behavior in MWCPPC	MWCPPC preferred	MrCpp's definition
-export_list filename	ignored	none	specify the file which contains a list of the shared library's exported entry points
-fp_contract on	same	same	generate multiply-add-fused instructions (default)
-maf on	same	-fp_contract on	generate multiply-add-fused instructions (default)
-fp_contract off	same	same	don't generate multiply-add-fused instructions
-maf off	same	-fp_contract off	don't generate multiply-add-fused instructions
-i pathname [,pathname…]	same	same	search for include files in directory specified by pathname
-inclpath normal | std | standard	same	default	treat <...> includes as system includes
-inclpath ignoresys | ignoresystem | nosys | nosystem	same	-nosyspath	treat <...> includes exactly like "..." includes
-inline on | all	same	same	functions are inlined as defined in the C++ source (default)
-inline none | off	same	same	inlining of C++ function expansions will be suppressed

MrC/MrCpp options	Behavior in MWCPPC	MWCPPC preferred	MrCpp's definition
-inline 0 | 1 | 2 | 3 | 4 | 5	turns inline off if 0 or on if 1-5	none	sets complexity limit on functions to be inlined.
-j0	ignored	none	accept two byte sequences for Japanese
-j1	ignored	none	accept two byte sequences for Taiwanese or Chinese
-j2	ignored	none	accept two byte sequences for Korean
-l filename	ignored	none	generates a listing file to the specified file with line numbers
-ldsize 64 | 128	only support 64; warning for 128	same	determines the size of long doubles (default = 64)
-load[c] filename	same	-prefix filename	restore the state of the compilation from the specified file
-noMapCR	same	same	changes the default mapping of \n to 0x0A and \r to 0x0D
-notOnce	same	same	allow multiple copies of header files
-o filename	same	same	specify output filename or directory
-opt off	same	same	suppress all optimizations (default with -sym on)
-opt local	same	-opt on	perform local optimizations (default with -sym off)
-opt size	see above	-opt size	optimize for size
-opt speed	see above	-opt all	optimize for speed
-p	same	same	output progress information to standard output

MrC/MrCpp options	Behavior in MWCPPC	MWCPPC preferred	MrCpp's definition
-shared_lib_export on	ignored	none	create a shared library and create file that contains entry point list
-shared_lib_export off	ignored	none	don't create shared library (default)
-sym off	same	same	disable generation of symbolic debugging information (default)
-sym on	same	same	enable generation of symbolic debugging information
-sym full	same as full-paths	same	enable generation of symbolic debugging information
-sym full[,nolines]	same as on	none	omit source line information
-sym full[,notypes]	same as on	none	omit type information
-sym full[,novars]	same as on	none	omit variable information
-sym full[,altypes]	same as on	none	generate type information for all types (including unreferenced types)
-target ppc | powerpc	ignored	none	generate code for PowerPC architecture
-target power	ignored	none	generate code for Power architecture
-target 601	ignored	none	generate code for PowerPC 601
-target 603	ignored	none	generate code for PowerPC 603
-traceback	same	same	generate IBM RS/6000 traceback tables
-w off	same	same	suppress all warnings
-w [is]err[or]	same	-w iserror	treat warnings as errors

MrC/MrCpp options	Behavior in MWCPPC	MWCPPC preferred	MrCpp's definition
-w n1[,n2…]	ignored	-w on	suppress the specified warning messages
-x	same	same	continue past the normal maximum error limit (4)
-xa public	ignored	none	template expansions have public scope
-xa static	ignored	none	template expansions have static scope (default)
-xa extern	ignored	none	templates are not expanded in the current compilation unit
-xi spec	ignored	none	instantiate the specified template
-y directory	ignored	none	write compiler temporary and object files to directory (default is same as source file)

Link and MWLink68K linkers

Differences between Link and MWLink68K.

- MWLink68K ignores several Link options. The most commonly used options that MWLink68K ignores are for alignment, branch islands, and the various special diagnostic tools: `-l`. `-la`, `-lf`, `-uf`, and `-x`. The details are in Table 10.8.
- The `-sym on|full` options are quite different. See Table 10.8 for details.

Table 10.8 Comparing Link and MWLink68K

Link options	Behavior in MWLink68K	MWLink68K preferred	Link's definition
-ac alignment	ignored	none	align code modules to 'n' byte boundaries
-ad alignment	ignored	none	align data modules to 'n' byte boundaries
-br off	ignored	none	do not generate branch islands (default)
-br on	ignored	none	generate branch islands where needed
-br 020	ignored	none	generate 32-bit branch islands where needed
-c creator	same	same	set resourceFile creator
-d	same	same	suppress duplicate definition warnings
-da	same	same	desk accessory - add NULL to segment names
-f	ignored	none	allow FORTRAN-style common data
-l	ignored	none	write a location map to output
-la	ignored	none	-l, include anonymous symbols in location map
-lf	ignored	none	-l, include file and location of definitions
-m mainEntry	same except default is main	same	use mainEntry as main entry point
-ma name=alias	same	same	create an alias for module name

Link options	Behavior in MWLink68K	MWLink68K preferred	Link's definition
-map	same	-map file-name	generate "friendly" link map
-mf	same	same	use MultiFinder temporary memory if necessary
-nomf	same	same	don't use temporary memory (the default)
-model far	same but affects jump	same	process 32-bit load-time relocatable references tables only
-model near	same but affects jump	same	prohibit 32-bit load-time relocatable references tables only; default
-msg [no]dup	same	same	(suppress) warnings about duplicate symbols
-msg [no]multiple	ignored	none	(suppress) multiple undefined symbol reports
-msg [no]warn	same	same	(suppress) warning messages
-newerdeps	ignored	none	ignored—for compatibility with ILink
-o outputFile | directory	same; default is MWLink68K.out	same	write to outputFile (default: Link.Out)
-opt Off	ignored	none	disable Object Pascal optimizations (default)
-opt On	ignored	none	enable optimizations
-opt On [,Info]	ignored	none	write optimization information to diagnostics

Link options	Behavior in MWLink68K	MWLink68K preferred	Link's definition
-opt On [,Names]	ignored	none	include MacsBug symbols within SelectorProc modules
-opt NoBypass	ignored	none	enable optimizations, but always dispatch
-opt NoBypass [,Info]	ignored	none	write optimization information to diagnostics
-opt NoBypass [,Names]	ignored	none	include MacsBug symbols within SelectorProc modules
-p	same	same	write progress information to diagnostics
-pg size	ignored	none	set page size to “size”; max: 32K; min: 1024; default: 4096
-ra [seg]=attr [,attr…]	same	same	set segment resource attributes:
-rn	same	same	don't include resource names in outputFile
-rt type=id	same	same	set resource type and id
-sg newSeg[=old[,old]…]	same	same	merge old segments into new segment
-sn oldSeg=newSeg	same	same	change segment name oldSeg to newSeg
-srt	ignored; does this by default	default	sort global data by “near” and “far” references (default)
-srtsg all | segName	ignored	none	place items referenced through Jump Table low in code resource.
-state	ignored	none	ignored—for compatibility with ILink

Link options	Behavior in MWLink68K	MWLink68K preferred	Link's definition
-sym Off	same	same	disable symbolic output (default)
-sym On | Full	same; noLabels, nolines,notypes,and novars are ignored	-sym full	enable symbolic output, can be followed by:
-t type	same	same	set resource file type (default APPL)
-uf unrefFile	ignored	none	write list of unreferenced modules to unrefFile
-w	same	same	suppress warnings
-wrap	ignored; does this by default	default	when jump table space is exhausted, put excess jump table entries in global data space, if available
-x crossRefFile	ignored	none	write cross reference to crossRefFile

PPCLink and MWLinkPPC

There are some important differences between PPCLink and MWLinkPPC. The differences are shown in Table 10.9.

Be aware that:

- MWLinkPPC does not support output to XCOFF format. It uses a proprietary library format.
- MWLinkPPC does not allow you to export symbols on the command line. Use the `-@export file` option instead.
- MWLinkPPC does not support `-weak`. Instead, mark the entire library weak with either `-weakimport` or `-weaklib`.
- MWLinkPPC generates SYM files. PPCLink does not.

Table 10.9 Comparing PPCLink and MWLinkPPC

PPCLink options	Behavior in MWLinkPPC	MWLink PPC preferred	PPCLink's description
-c restype	same	same	set outputFile creator
-d	same	same	suppress duplicate symbol warnings
-dead on | off	Always on, unless building a library when it is off	same	control garbage collection of unused code and data. Default value depends on value of -xm flag as follows: -xm executable -dead on; -xm sharedLibrary -dead on; -xm library -dead off
-@export exportFile	same	same	export symbols contained in file, one symbol per line
-export name[,name]…	ignored	-@export filename	export named symbol(s)
-init initEntry	same	same	set initialization entry point
-initbefore name[,name]…	same	same	mark the specified shared libraries as 'init before'
-librename old=new	same	same	rename libraries (e.g. -librename InterfaceLib.xcoff=InterfaceLib)
-m[ain] mainEntry	same	same	set main entry point (default: __start)
-map mapFile	same	same	create a link map with the given file name
-mf	same; default	same	use MultiFinder memory
-nomf	same	same	don't use MultiFinder memory (default)

PPCLink options	Behavior in MWLinkPPC	MWLink PPC preferred	PPCLink's description
-o outputFile	same; default is MWLinkPPC.out	same	set outputFile name (default: PPCLink.out)
-outputformat default	ignored; output always PEF except for libraries, which use proprietary format	none	set outputFile format
-p	same	same	write progress information to diagnostics
-packdata on | off	ignored	none	compress the data section (default: on)
-roistext on | off	ignored	none	place read-only data in the .text section (default: off)
-share context	same	default	set the sharing mode for the data section (default: context) context = shared within a context
-share global	same	-sharedata	global = shared across all contexts
-share protected	ignored	none	protected = readable by all contexts, writable by privileged contexts
-sym on | off	same but CodeWarrior creates sym file and PPCLink puts it in the pef file	same	include or exclude debug symbols (default: off)
-sym big | nobig	same as on	-sym full	include or exclude debug symbols (default: off)
-t restype	same	same	set outputFile type

PPCLink options	Behavior in MWLinkPPC	MWLink PPC preferred	PPCLink's description
-term termEntry	same	same	set termination entry point
-tocdataref on | off	ignored	none	optimize runtime calculations of data addresses (default: off)
-uf filename	ignored	none	report unused code and data symbols to filename
-unmangle on | off	ignored	none	unmangle C++ symbols in listings (default: on)
-v	same	-p	use verbose progress (implies -p)
-vercur n	same	-cv n	set PEF container 'current version' to n (default: 0)
-verdef n	same	-dv n	set PEF container 'definition version' to n (default: 0)
-verimp n	same	-uv n	set PEF container 'implementation version' to n (default: 0)
-w[arn]	same	same	suppress warning messages
-weak name[,name]…	ignored	none	mark specified imports as 'weak'
-weaklib name[,name]…	same	same	mark specified shared libraries as 'weak'
-xm e[xecutable]	same	-xm a	generate an application (default)
-xm l[ibrary]	same	same	generate a static library
-xm s[haredlibrary]	same	same	generate a shared library

11

CodeWarrior IDE and Command-Line Tools Compared

This chapter gives an option-by-option comparison of the CodeWarrior command line and IDE compilers and linkers. This information would be useful when translating a Mac OS CodeWarrior IDE project to another environment such as MPW or ToolServer, or when working with the BeBox platform in a command line environment.

Comparison Overview

When you work in the various CodeWarrior IDE settings panels, you can set a wide variety of options that affect the compilers and linkers. In this chapter we present a series of tables that show the relationship between the IDE settings panel options and the CodeWarrior command line tools.

This chapter is organized by IDE settings panel. The major sections in this chapter are:

- Target Settings—68K, PowerPC, and access paths
- Language Settings—C/C++ and Pascal language settings and warnings
- Code Generation—68K, PowerPC, and disassembler
- Linker—68K, CFM68K, and PowerPC/PEF

In all cases, text in braces—the [and] characters—is optional. The braces themselves never appear in the actual command. For example, you could write `-str[ings] pool` as either `-strings pool` or `-str pool`.

Target Settings

There are two different kinds of projects settings under CodeWarrior that can be changed.

- 68K Target
- PPC Target

In addition, this section shows how to setup the paths along which a compiler will search for files.

- Access Paths

68K Target

Figure 11.1 shows the 68K settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a new 68K project. Selecting **68K Target** causes options to appear in the settings panel as shown in Figure 11.2.

Figure 11.1 IDE Settings Panel Options (68K Projects)

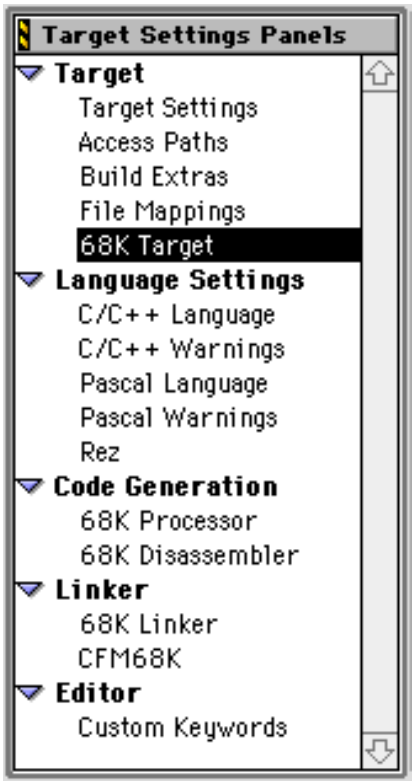
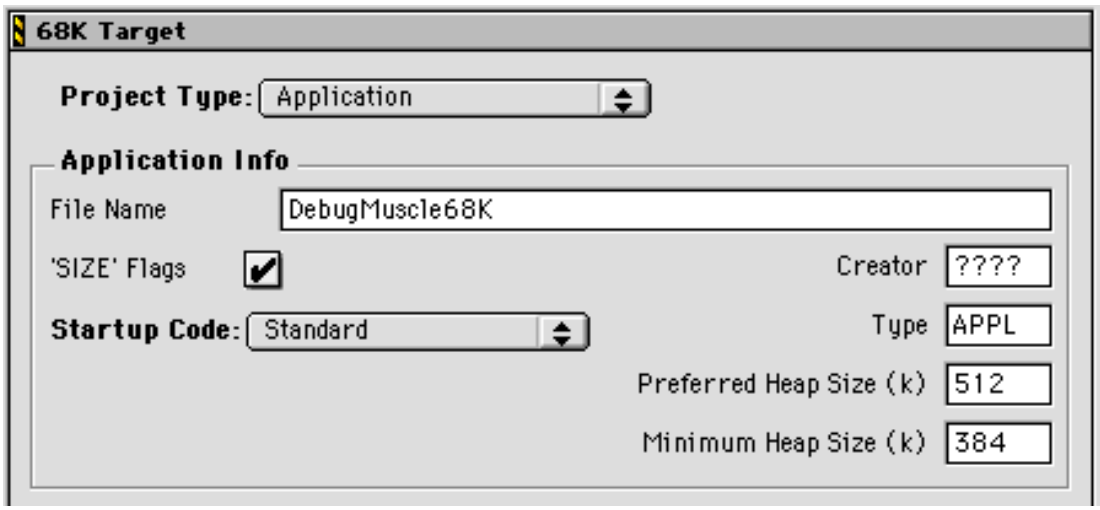


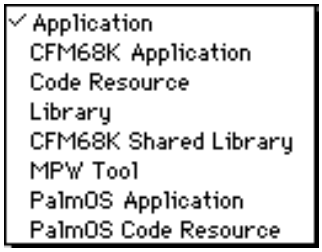
Figure 11.2 68K Target Settings Panel



There are several different kinds of 68K projects that can be selected with the Project Type popup menu shown in Figure 11.3. The types of 68K projects are:

- 68K Application Projects
- CFM68K Application Projects
- 68K Code Resource Projects
- 68K Library Projects
- CFM68K Shared Library Projects
- 68K MPW Tool Projects
- PalmOS Application Projects
- PalmOS Code Resource Projects

Figure 11.3 Project Type Popup Menu (68K Project)



68K Application Projects

Figure 11.2 shows the settings panel for a 68K application project. Figure 11.4 shows additional popup menu options that are available in this settings panel.

Table 11.1 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.4 'SIZE' Flags and Startup Code Popup Menus

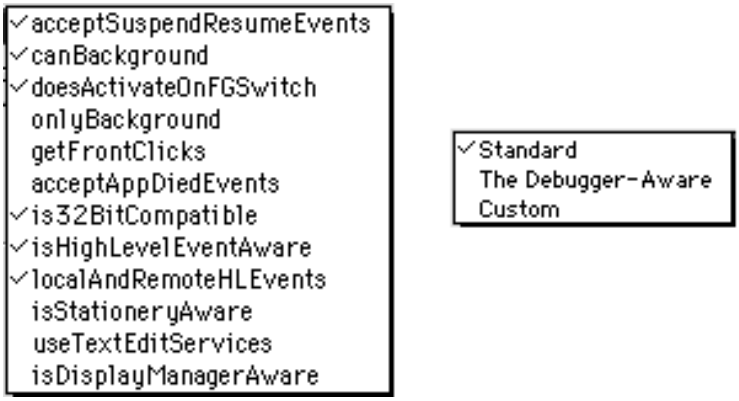


Table 11.1 68K Applications: MWLink68K Tool Command Line

Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	Application	-xm a[pplication]
File Name		-o filename | directory specify output file name or directory
'SIZE' flags	acceptSuspend-ResumeEvents	-sizeflags suspendevents
'SIZE' flags	canBackground	-sizeflags canbackground
'SIZE' flags	doesActivateOnFGSwitch	-sizeflags multifinderaware
'SIZE' flags	onlyBackground	-sizeflags onlybackground
'SIZE' flags	getFrontClicks	-sizeflags getfrontclicks
'SIZE' flags	acceptAppDiedEvents	-sizeflags childdiedevents
'SIZE' flags	is32BitCompatible	-sizeflags 32bitclean
'SIZE' flags	isHighLevelEventAware	-sizeflags highlevelevents
'SIZE' flags	localAndRemoteHLEv-ents	-sizeflags remoteevents
'SIZE' flags	isStationeryAware	-sizeflags stationary
'SIZE' flags	useTextEditServices	-sizeflags textedit
'SIZE' flags	isDisplayManagerAware	-sizeflags displayaware
Startup Code	Standard	This is the default unless -sym full or -tdb are specified
Startup Code	The Debugger-Aware	-sym full or use -tdb to generate linker support for Jasik without the .SYM file being generated. Note that -sym full must also be specified on the compiler command line.
Startup Code	Custom	-custom
Creator		-c creator, default is '????'

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Type		-t APPL for an application, or could be any other valid type such as appe (background-only application) APPL is the default if not specified
Preferred Heap Size (k)		-sizemax num (default is 384)
Minimum HeapSize (k)		-sizemin num (default is 384)

CFM68K Application Projects

Figure 11.5 shows the settings panel for a CFM68K application project. Figure 11.6 shows additional popup menu options that are available in this settings panel.

Table 11.2 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.5 CFM68K Application Target Settings Panel

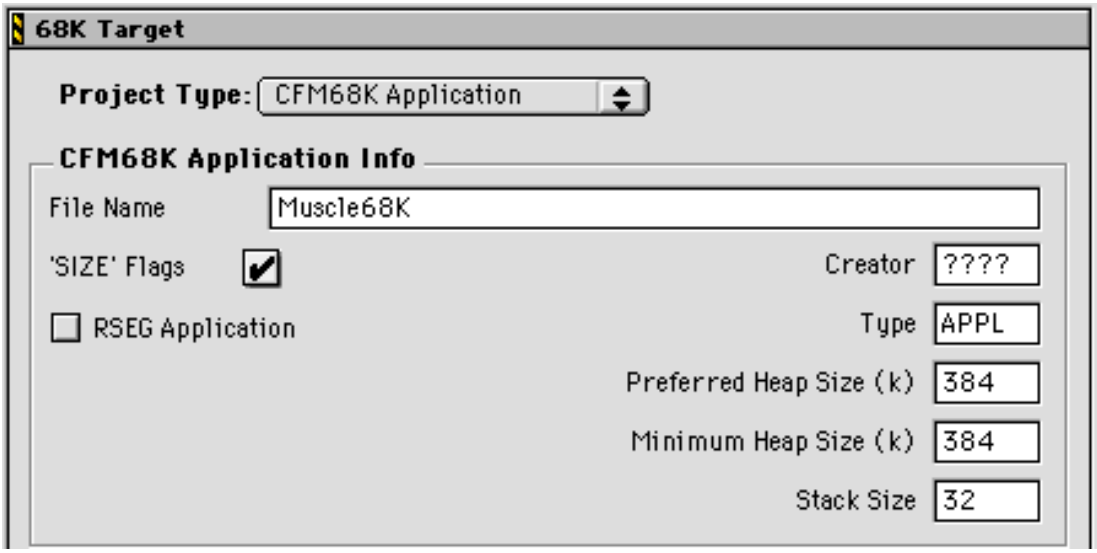


Figure 11.6 'SIZE' Flags Popup Menu

Table 11.2 CFM68K Application: MWLink68K Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	CFM68K Application	-xm a[pplication] -model CFMflatdf
File Name		-o filename | directory specify output file name or directory
'SIZE' flags	acceptSuspend-ResumeEvents	-sizeflags suspendevents
'SIZE' flags	canBackground	-sizeflags canbackground
'SIZE' flags	doesActivateOnFGSwitch	-sizeflags multifinderaware
'SIZE' flags	onlyBackground	-sizeflags onlybackground
'SIZE' flags	getFrontClicks	-sizeflags getfrontclicks
'SIZE' flags	acceptAppDiedEvents	-sizeflags childdiedevents
'SIZE' flags	is32BitCompatible	-sizeflags 32bitclean
'SIZE' flags	isHighLevelEventAware	-sizeflags highlevelevents
'SIZE' flags	localAndRemoteHLEv-ents	-sizeflags remoteevents

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
'SIZE' flags	isStationeryAware	-sizeflags stationary
'SIZE' flags	useTextEditServices	-sizeflags textedit
'SIZE' flags	isDisplayManagerAware	-sizeflags displayaware
RSEG Application		Information not available at press time. Please check the release notes for the latest information.
Creator		-c creator, default is '????'
Type		-t APPL for an application, or could be any other valid type such as appe (background-only application) APPL is the default if not specified
Preferred Heap Size (k)		-sizemax num (default is 384)
Minimum HeapSize (k)		-sizemin num (default is 384)
Stack Size (k)		-stacksize num (default is kDefaultStackSize)

68K Code Resource Projects

Figure 11.7 shows the settings panel for a 68K code resource project. Figure 11.8 shows additional popup menu options that are available in this settings panel.

Table 11.3 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.7 68K Code Resource Target Settings Panel

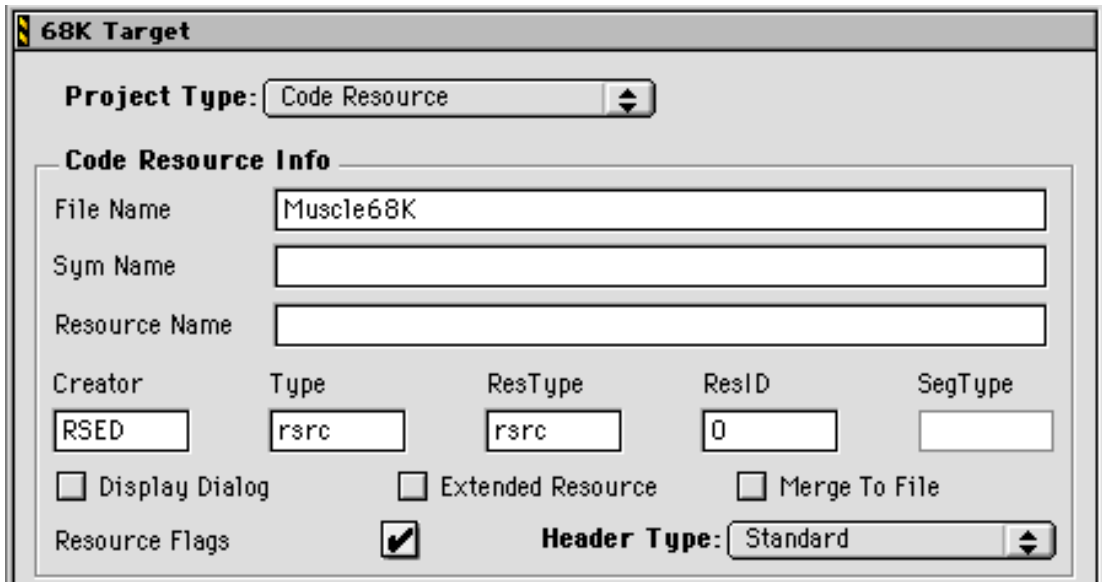


Figure 11.8 Resource Flags and Header Popup Menus

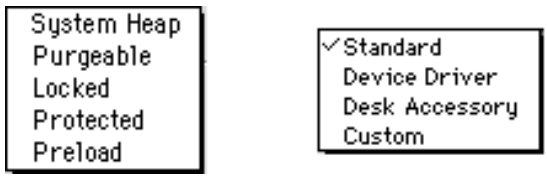


Table 11.3 Code Resource: MWLink68K Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	Code Resource	-xm c[oderesource]
File Name		-o filename | directory specify output file name or directory
Sym Name		-osym filename (default is objfilename.SYM)

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Creator		-c creator, default is '????'
Type		-t rsrc for a CODE resource, or could be any other appropriate type
Resource Type and ResID		-rt type=id e.g. -rt PROC=128 would create a code resource of type PROC and id number 128
SegType		-rsrcsegtype type Default is first 3 characters of Resource Type + 'c'. -rsrcname is used to set the type of the first segment
Display Dialog		In order to specify where to put the code resource (whether merging or generating a new file) use the -o option.
Extended Resource		-rsrcfar (Default is no extended resource)
Merge To File		-merge on
Resource Name		-rsrcname name Name is a ResEdit name, "" is default
Resource Flags	System Heap	-ra resSysHeap
Resource Flags	Purgeable	-ra resPurgeable
Resource Flags	Locked	-ra resLocked
Resource Flags	Protected	-ra resProtected
Resource Flags	Preload	-ra resPreload

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Header Type	Standard	This is the default when -xm c[oderesource] is specified on the command line
Header Type	Device Driver	-drvr Name the driver with -rsrcname (a leading period is NOT generated by the linker).
Header Type	Desk Accessory	-da Name the desk accessory with -rsrcname (a leading NULL is generated by the linker)
Header Type	Custom	-custom

68K Library Projects

Figure 11.9 shows the settings panel for a 68K library project.

Table 11.4 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.9 68K Library Target Settings Panel

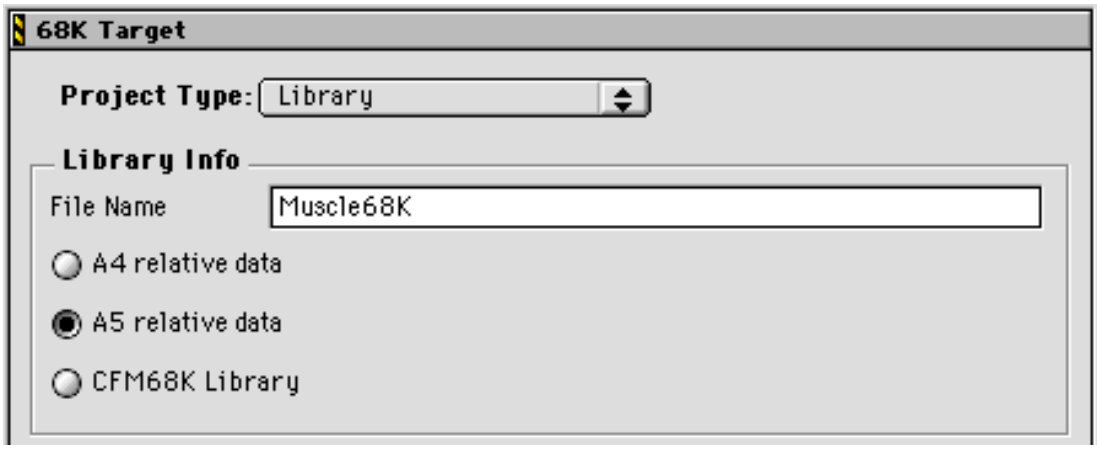


Table 11.4 68K Library: MWLink68K Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	Library	-xm l[ibrary] or -xm libraryA4 (see table entries below for relative data)
File Name		-o filename | directory specify output file name or directory
A4 relative data		-xm libraryA4
A5 relative data		-xm l[ibrary]
CFM68K Library		-model CFMflat -xm l[ibrary]

CFM68K Shared Library Projects

Figure 11.10 shows the settings panel for a CFM68K shared library project.

Table 11.5 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.10 CFM68K Shared Library Target Settings Panel

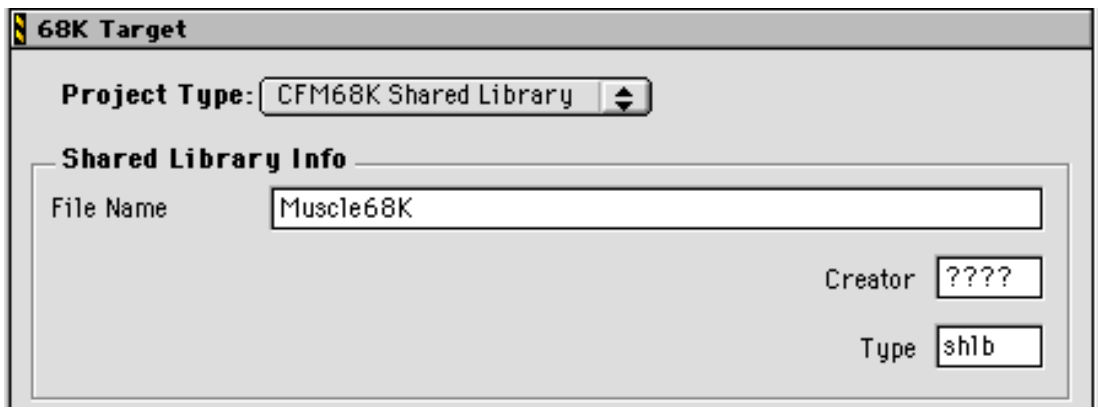


Table 11.5 CFM68K Shared Library: MWLink68K Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	CFM68K Shared Library	-xm s[haredLibrary]
File Name		-o filename | directory specify output file name or directory
Creator		-c creator, default is '????'
Type		-t shlb is a customary type for a shared library

68K MPW Tool Projects

Figure 11.11 shows the settings panel for a 68K MPW Tool project.

Table 11.6 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.11 68K MPW Tool Target Settings Panel

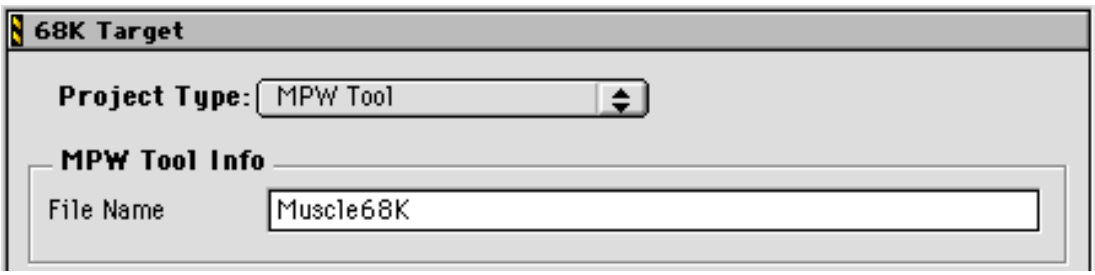


Table 11.6 68K MPW Tools: MWLink68K Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	MPW Tool	-xm m[pwtool]
File Name		-o filename | directory specify output file name or directory

PalmOS Application Projects

Figure 11.12 shows the settings panel for a 68K PalmOS Application project. Table 11.7 shows the command-line options that correspond to IDE settings.

Figure 11.12 PalmOS Application Options Panel

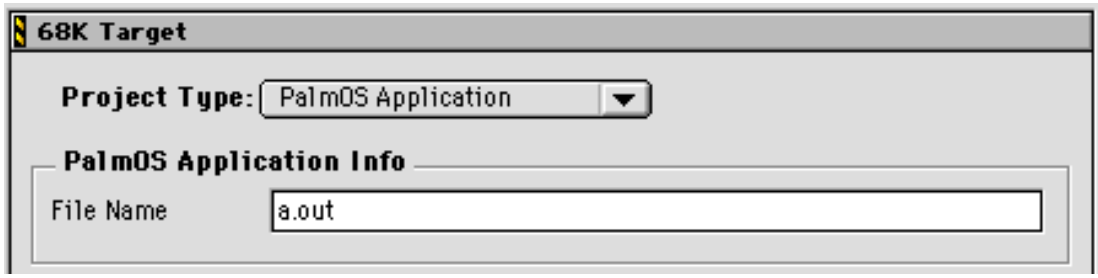


Table 11.7 68K PalmOS Application: MWLink68K Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	PalmOS Application	-xm a[pplication]
File Name		-o filename | directory specify output file name or directory

PalmOS Code Resource Projects

Figure 11.13 shows the settings panel for a 68K PalmOS Application project. Table 11.7 shows the command-line options that correspond to IDE settings.

Figure 11.13 PalmOS Code Resource Options Panel

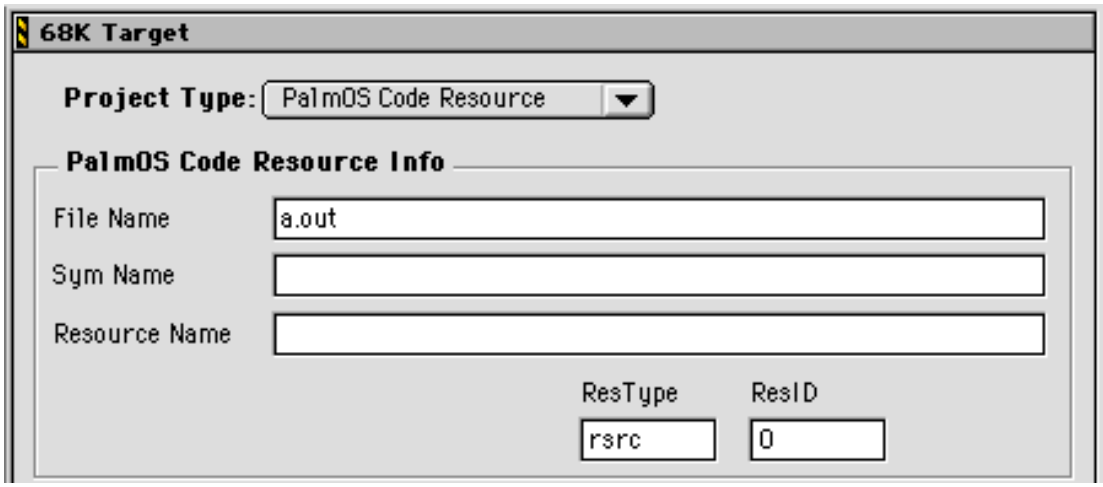


Table 11.8 Code Resource: MWLink68K Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	PalmOS Code Resource	-xm c[oderesource]
File Name		-o filename | directory specify output file name or directory
Sym Name		-osym filename (default is objfilename.SYM)

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Resource Type and ResID		-rt type=id e.g. -rt PROC=128 would create a code resource of type PROC and id number 128
Resource Name		-rsrcname name Name is a ResEdit name, “” is default

PPC Target

Figure 11.14 shows the PPC Target panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a PPC project. Selecting **PPC Target** causes options to appear in the settings panel as shown in Figure 11.15.

Figure 11.14 IDE Settings Panel Options (PPC Projects)

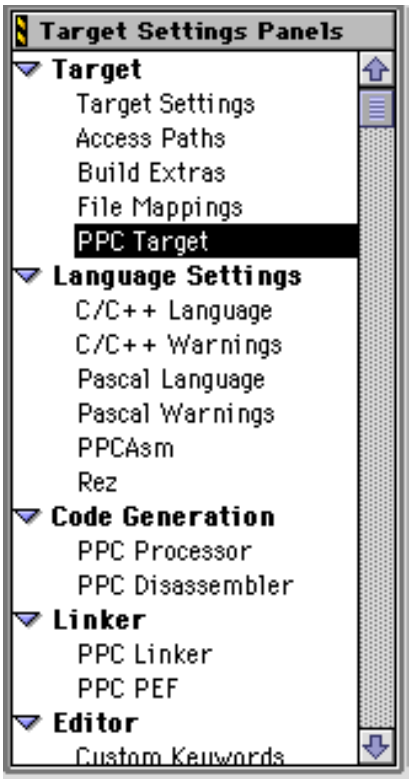
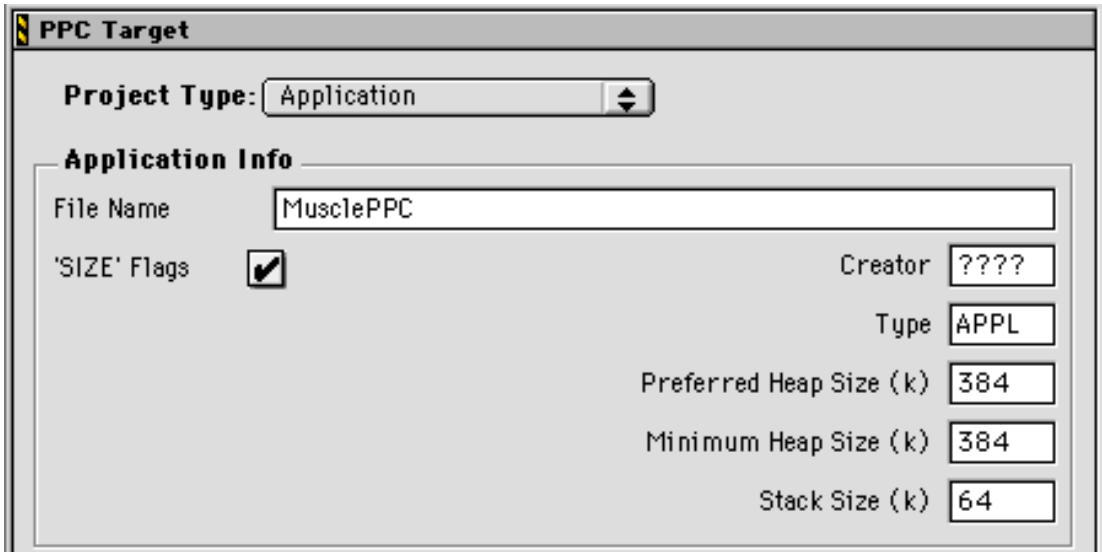


Figure 11.15 PPC Target Settings Panel



There are several different kinds of PPC projects that can be selected with the Project Type popup menu as shown in Figure 11.16.

- PPC Application Projects
- PPC Shared Library Projects
- PPC Code Resource Projects
- PPC Library Projects

Figure 11.16 PPC Project Type Popup Menu

PPC Application Projects

Figure 11.15 shows the settings panel for a PPC application project. Figure 11.16 shows additional popup menu options that are available in this settings panel.

Table 11.9 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.17 'SIZE' Flags Popup Menu

Table 11.9 PPC Application: MWLinkPPC (Mac OS) and mwcc/mwld (BeOS) Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	Application	-xm a[pplication]
File Name		-o filename | directory Specify output file name or directory (Mac OS) Write object code to filename or directory if used with -c, otherwise passed to linker (BeOS)
'SIZE' flags	acceptSuspend-ResumeEvents	-sizeflags suspendevents (Mac OS only)
'SIZE' flags	canBackground	-sizeflags canbackground (Mac OS only)

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
'SIZE' flags	doesActivateOnFGSwitch	-sizeflags multifinderaware (Mac OS only)
'SIZE' flags	onlyBackground	-sizeflags onlybackground (Mac OS only)
'SIZE' flags	getFrontClicks	-sizeflags getfrontclicks (Mac OS only)
'SIZE' flags	acceptAppDiedEvents	-sizeflags childdiedevents (Mac OS only)
'SIZE' flags	is32BitCompatible	-sizeflags 32bitclean (Mac OS only)
'SIZE' flags	isHighLevelEventAware	-sizeflags highlevelevents (Mac OS only)
'SIZE' flags	localAndRemoteHLEv-ents	-sizeflags remoteevents (Mac OS only)
'SIZE' flags	isStationeryAware	-sizeflags stationary (Mac OS only)
'SIZE' flags	useTextEditServices	-sizeflags textedit (Mac OS only)
'SIZE' flags	isDisplayManagerAware	-sizeflags displayaware (Mac OS only)
Creator		-c creator, default is '????' (Mac OS) -fc filecreator, default is '????' (BeOS)
Type		-t APPL for an application, or could be any other valid type such as appe, a background-only application. APPL is the default if not specified. (Mac OS) -ft filetype 'BAPP' is default (BeOS)

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Preferred Heap Size (k)		-sizemax num Default for num is 384 (Mac OS only)
Minimum HeapSize (k)		-sizemin num Default for num is 384 (Mac OS only)
Stack Size (k)		-stacksize num Default for num is kDefaultStackSize (Mac OS only)

PPC Shared Library Projects

Figure 11.18 shows the settings panel for a PPC shared library project.

Table 11.10 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.18 PPC Shared Library Target Settings Panel

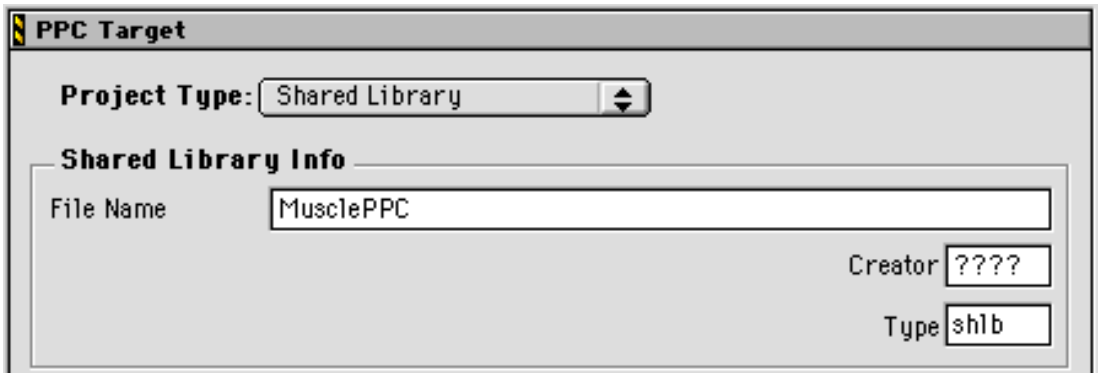


Table 11.10 Shared Library: MWLinkPPC (Mac OS) and mwcc/

mwld (BeOS) Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	Shared Library	-xm s[haredLibrary] (Mac OS or BeOS) -G (BeOS only)
File Name		-o filename | directory Specify output file name or directory (Mac OS) Write object code to filename or directory if used with -c, otherwise passed to linker (BeOS)
Creator		-c creator, default is '????' (Mac OS) -fc filecreator, default is '????' (BeOS)
Type		-t filetype (Mac OS) -ft filetype (BeOS) shlb is a customary filetype for a shared library for both Mac OS and BeOS

PPC Code Resource Projects

Figure 11.19 shows the settings panel for a PPC code resource project. Figure 11.20 shows additional popup menu options that are available in this settings panel.

Table 11.11 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.19 PPC Code Resource Target Settings Panel

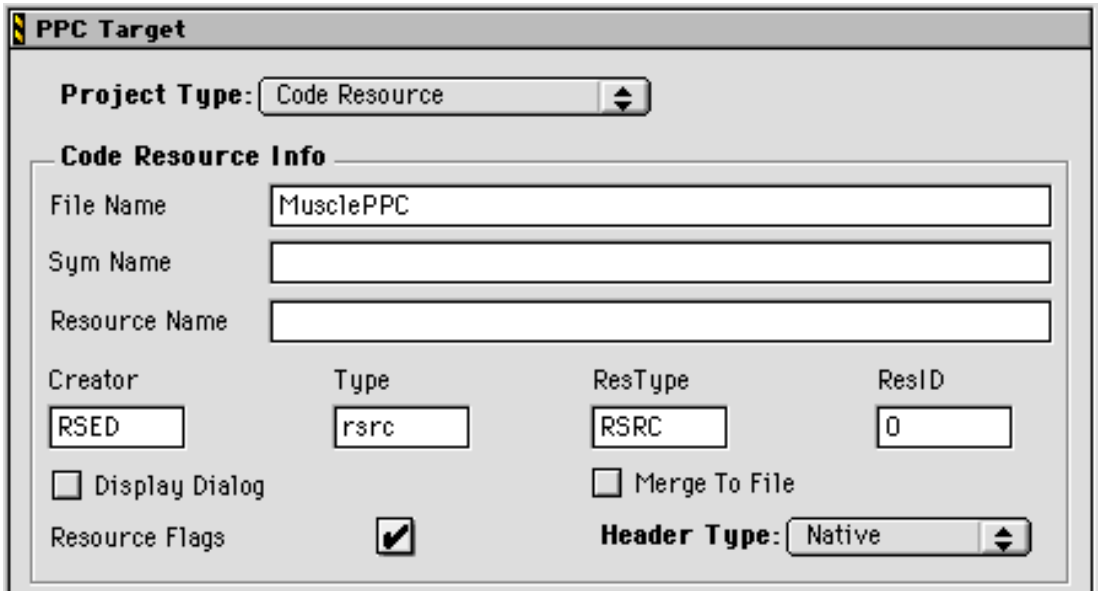


Figure 11.20 Resource Flags and Header Popup Menus

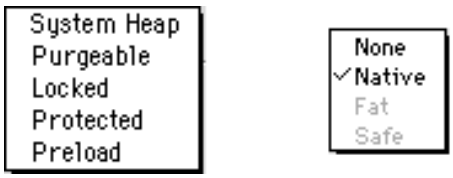


Table 11.11 Code Resource: MWLinkPPC Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	Code Resource	-xm c[oderesource]
File Name		-o filename | directory specify output file name or directory

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Sym Name		-osym filename (default is objfilename.xSYM)
Creator		-c creator, default is '????'
Type		-t rsrc for a CODE resource, or could be any other appropriate type
Resource Type and ResID		-rt type=id e.g. -rt PROC=128 would create a code resource of type PROC and id number 128
Display Dialog		In order to specify where to put the code resource (whether merging or generating a new file) use the -o option.
Merge To File		-merge on
Resource Name		-rsrcname name name is a ResEdit name, "" is default
Resource Flags	System Heap	-ra resSysHeap
Resource Flags	Purgeable	-ra resPurgeable
Resource Flags	Locked	-ra resLocked
Resource Flags	Protected	-ra resProtected
Resource Flags	Preload	-ra resPreload
Header Type	None	This is the default when -xm c[oderesource] is specified on the command line.
Header Type	Native	-native

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Header Type	Fat	-fat filename (filename should be for a resource that matches the current project's resType and ResID found in filename, and should already contain a 68K code resource)
Header Type	Safe	-safe filename (filename should be for a resource that matches the current project's resType and ResID found in filename, and should already contain a 68K code resource)

PPC Library Projects

Figure 11.21 shows the settings panel for a PPC library project.

Table 11.12 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.21 PPC Library Target Settings Panel

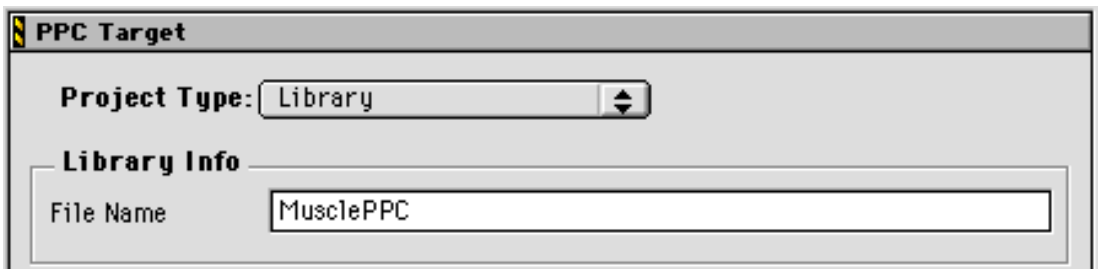


Table 11.12 PPC Library: MWLinkPPC (Mac OS) and mwcc/mwld

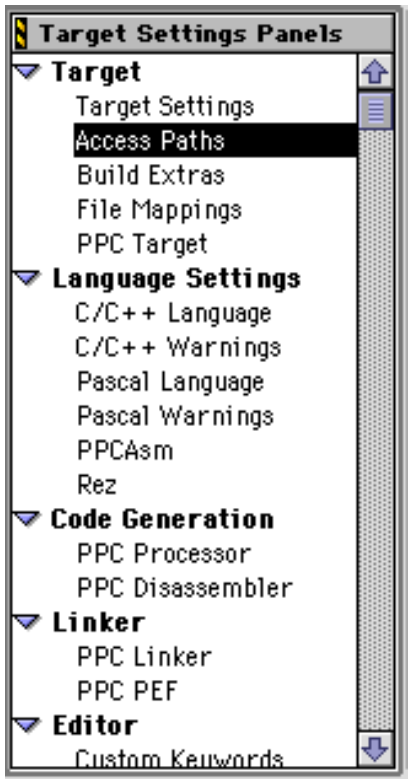
(BeOS) Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Project Type	Library	-xm l[ibrary]
File Name		-o filename | directory Specify output file name or directory (Mac OS) Write object code to filename or directory if used with -c, otherwise passed to linker (BeOS)

Access Paths

Figure 11.22 shows the 68K settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a 68K or PPC project. Selecting **Access Paths** causes options to appear in the settings panel as shown in Figure 11.23.

Figure 11.22 IDE Settings Panel (Access Paths)



This feature is used by the compiler as the method for specifying additional non-paths to search when looking for files. The command line compilers use the -i option to specify these paths to files that may be needed at compile time.

An example of using this feature for a 68K project would be:

`MWC68K [options…] -i “HD:Dev:” -i “HD:Core:”` filename.cpp

Note that the MPW environment uses a colon “:” as the directory delimiter, while the the BeOS development environment uses a forward slash “/.”

Both the BeOS and Mac OS have an additional option -i-. This option causes all include directories specified after this option to be searched as if they are system header files, like <includefile.h>.

The -cwd option for all the compiler tools specifies the first access path to be searched, as detailed in Table 11.13.

Figure 11.23 Access Paths Settings Panel

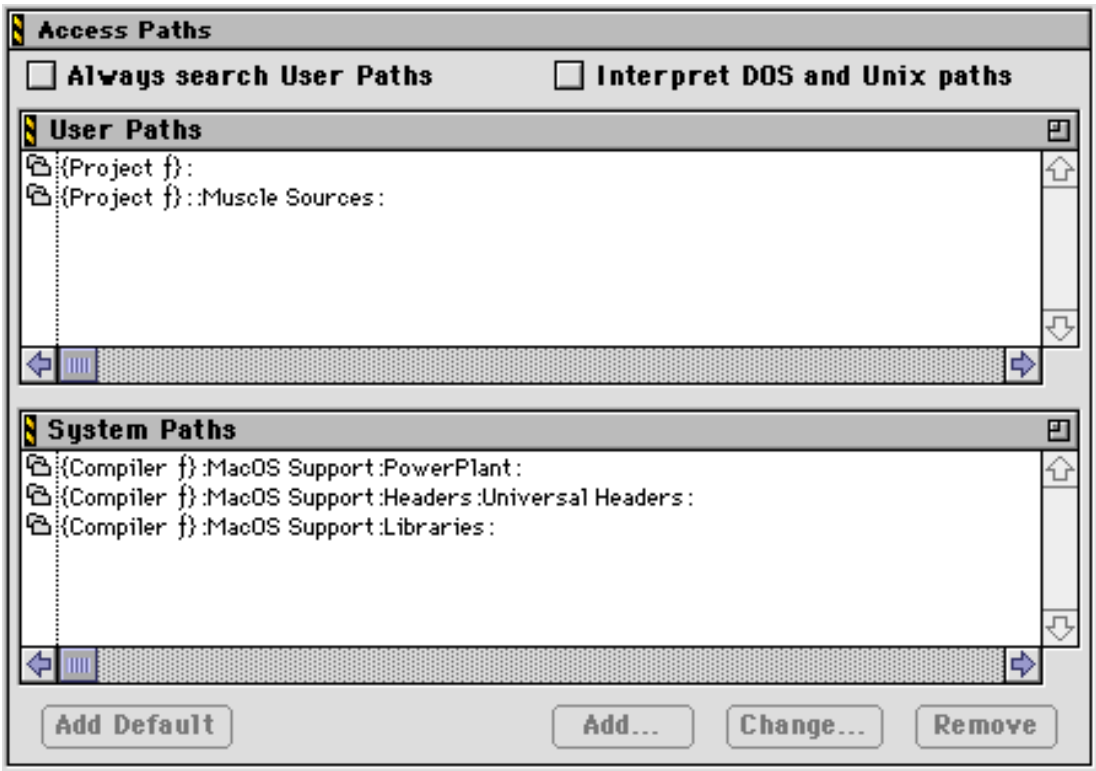


Table 11.13 Possible Usages of the -cwd Compiler Option

Command Line Option	Description
-cwd proj	Begin search in current working directory first

Command Line Option	Description
-cwd source	Begin search in current source file's directory
-cwd explicit	Begin searching in the first directory specified with -i (requires a -i specifier on the command line)

Language Settings

The CodeWarrior IDE allows different C/C++ compiler language parameters to be customized.

- C/C++ Language Settings
- C/C++ Warnings
- Pascal Language Settings
- Pascal Warnings
- PPCAsm
- Rez (and DeRez)

C/C++ Language Settings

Figure 11.24 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a 68K or PPC project. Selecting **C/C++ Language** causes options to appear in the settings panel as shown in Figure 11.25.

Figure 11.26 shows additional popup menu selections for these settings. Table 11.14 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.24 IDE Settings Panel (C/C++ Language)

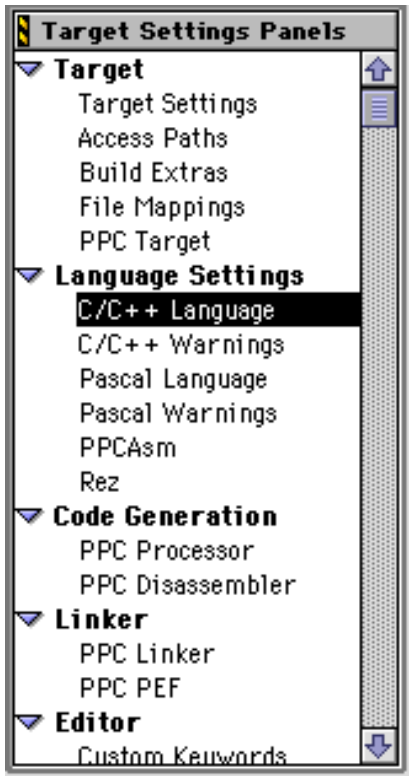


Figure 11.25 C/C++ Language Settings Panel

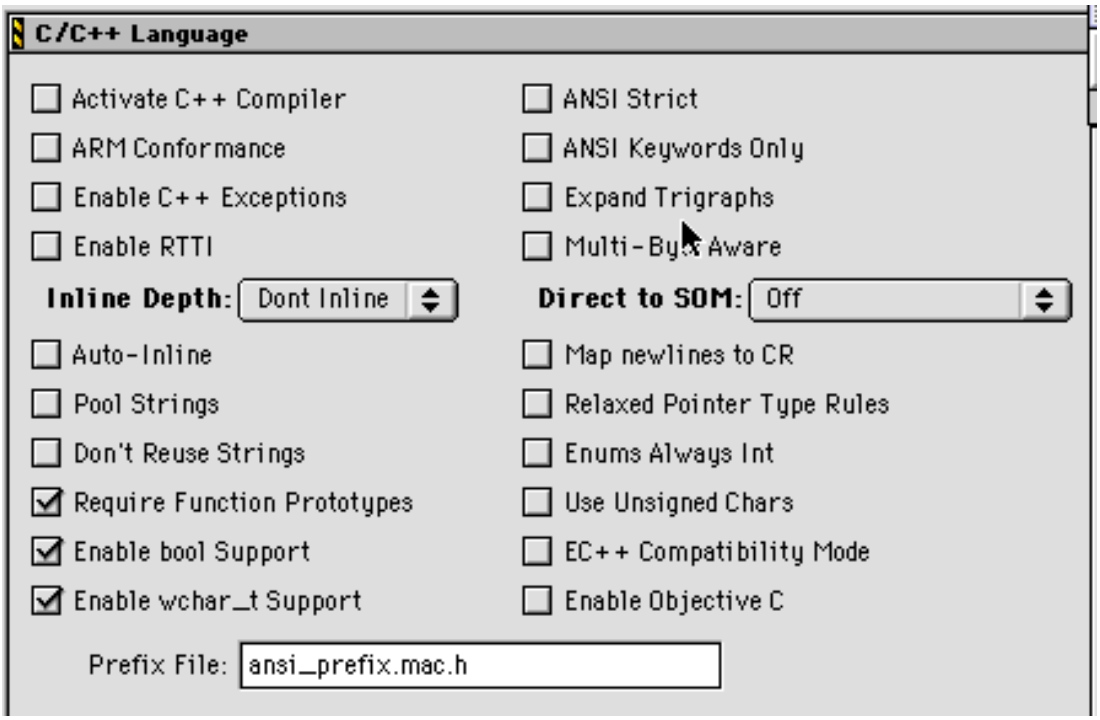


Figure 11.26 C/C++ Language Source Model and Inlining Popup Menus

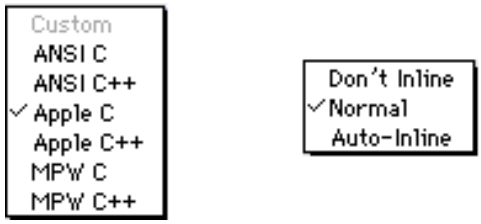


Table 11.14 MWCPPC, mwcc/mwld and MWC68K Command Line

Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Source Model	Custom	This option requires the user to specify which model options are desired
Source Model	ANSI C	-strict on -appleext off -trigraphs on -Cpp_exceptions off -dialect C -r (Mac OS and BeOS)
Source Model	ANSI C++	-strict on -appleext off -trigraphs on -RTTI on -dialect Cplus -r (Mac OS) -strict on -appleext on -trigraphs on -RTTI on -dialect Cplus -r (BeOS)
Source Model	Apple C	-Cpp_exceptions off -dialect C -r (Mac OS only)
Source Model	Apple C++	-dialect Cplus -r -RTTI on (Mac OS only)
Source Model	MPW C	-Cpp_exceptions off -dialect C -r -enums int -mpw_chars (Mac OS only)
Source Model	MPW C++	-dialect Cplus -r -RTTI on -enums int (Mac OS only)
Activate C++ Compiler		-dialect C or Cplus (Mac OS and BeOS) Default is C
ARM Conformance		-ARM on or off (Mac OS and BeOS) Default is off
Enable C++ Exceptions		-Cpp_exceptions on or off (Mac OS and BeOS) Default is on
Enable RTTI		-RTTI, Default is off (Mac OS) -RTTI on or off, Default is on (BeOS)
Inlining	Don't Inline	-inline off (Mac OS and BeOS)
Inlining	Normal	-inline on, Default is on (Mac OS and BeOS)
Inlining	Auto-Inline	-inline auto (Mac OS and BeOS)
Pool Strings		-str[ings] pool (Mac OS and BeOS)

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Don't Reuse Strings		-str[ings] dontreuse (Mac OS and BeOS)
Enable bool Support		-bool on or off
Enable wchar_t Support		`-wchar_t on|off`—enable wchar_t as a built-in C++ type
Require Function Prototypes		-r[equireprotos] (Mac OS and BeOS)
ANSI Strict		-strict on (Mac OS and BeOS, this is not the same thing as -ansi strict)
ANSI Keywords Only		-appleext off (Mac OS and BeOS) Default is on
Expand Trigraphs		-trigraphs on or off (Mac OS and BeOS) Default is off
Multi-Byte Aware		-j0 (Mac OS only)
MPW Newlines		-mapcr (Mac OS only)
MPW Pointer Type Rules		-relax_pointers (Mac OS C Language only)
Enums Always Int		-enum int (Mac OS and BeOS) This option should always be used for BeOS
Use Unsigned Chars		-char unsigned (Mac OS and BeOS) Default is signed
EC++ Embedded Compatibility		`-ec++ on|off`—enable Embedded C++, this disables some C++ features such as templates and exceptions

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Enable Objective C		`-objc on|off`—enable Objective-C language extensions
Prefix File		-prefix filename (Mac OS and BeOS)

C/C++ Warnings

Figure 11.27 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a 68K or PPC project. Selecting **C/C++ Warnings** causes options to appear in the settings panel as shown in Figure 11.28.

Table 11.15 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.27 IDE Settings Panel (C/C++ Warnings)

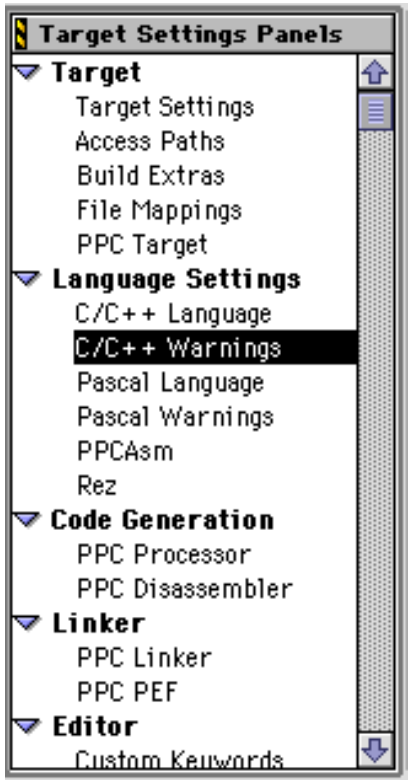

Target Settings Panels
Target
Target Settings
Access Paths
Build Extras
File Mappings
PPC Target
Language Settings
C/C++ Language
C/C++ Warnings
Pascal Language
Pascal Warnings
PPCAsm
Rez
Code Generation
PPC Processor
PPC Disassembler
Linker
PPC Linker
PPC PEF
Editor
Custom Keywords

Figure 11.28 C/C++ Warnings

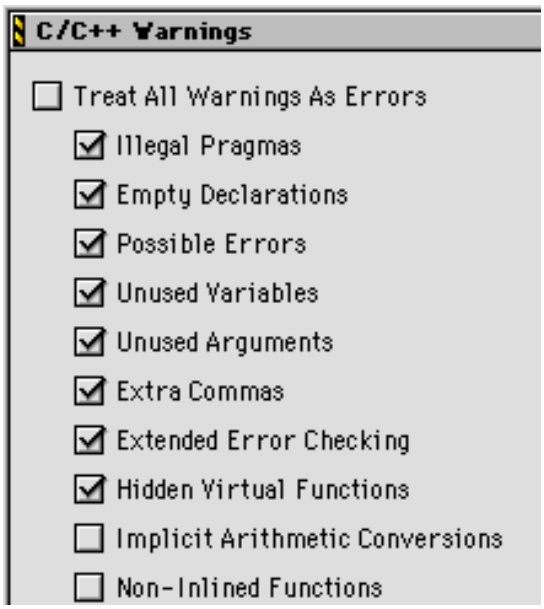


Table 11.15 C/C++ Warnings: MWC68K, MWCPPC (Mac OS) and mwcc/mwld (BeOS) Command Line Options

NOTE: -w[arnings] on turns on all warnings except -w[arnings] [is]err[or], and is the default (Mac OS and BeOS).

IDE Option Name	Command Line Option(s)
Treat All Warnings As Errors	-w[arnings] [is]err[or]
Illegal Pragmas	-w[arnings] [no]pragmas (Mac OS and BeOS)
Empty Declarations	-w[arnings] [no]emptydecl (Mac OS and BeOS)
Possible Errors	-w[arnings] [no]possible (Mac OS and BeOS)
Unused Variables	-w[arnings] [no]unusedvar (Mac OS and BeOS)
Unused Arguments	-w[arnings] [no]unusedarg (Mac OS and BeOS)
Extra Commas	-w[arnings] [no]extracomma (Mac OS and BeOS)

IDE Option Name	Command Line Option(s)
Extended Error Checking	-w[arnings] [no]extended (Mac OS and BeOS)
Hidden Virtual Functions	-w[arnings] [no]hidevirtual

Pascal Language Settings

Figure 11.29 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a 68K or PPC project. Selecting **Pascal Language** causes options to appear in the settings panel as shown in Figure 11.30.

Table 11.16 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.29 IDE Settings Panel (Pascal Language)

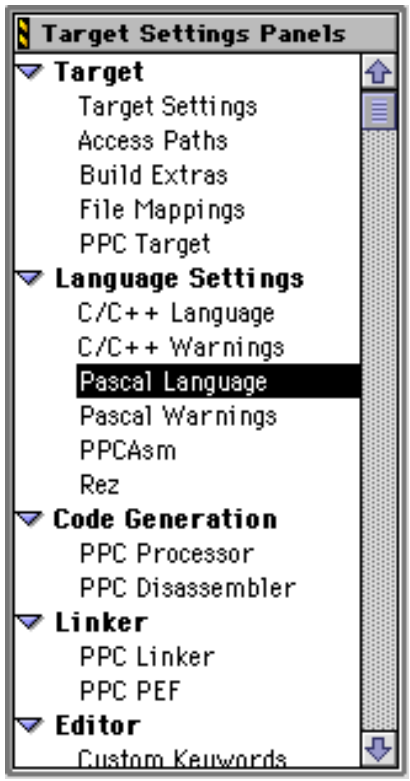


Figure 11.30 Pascal Language Settings Panel

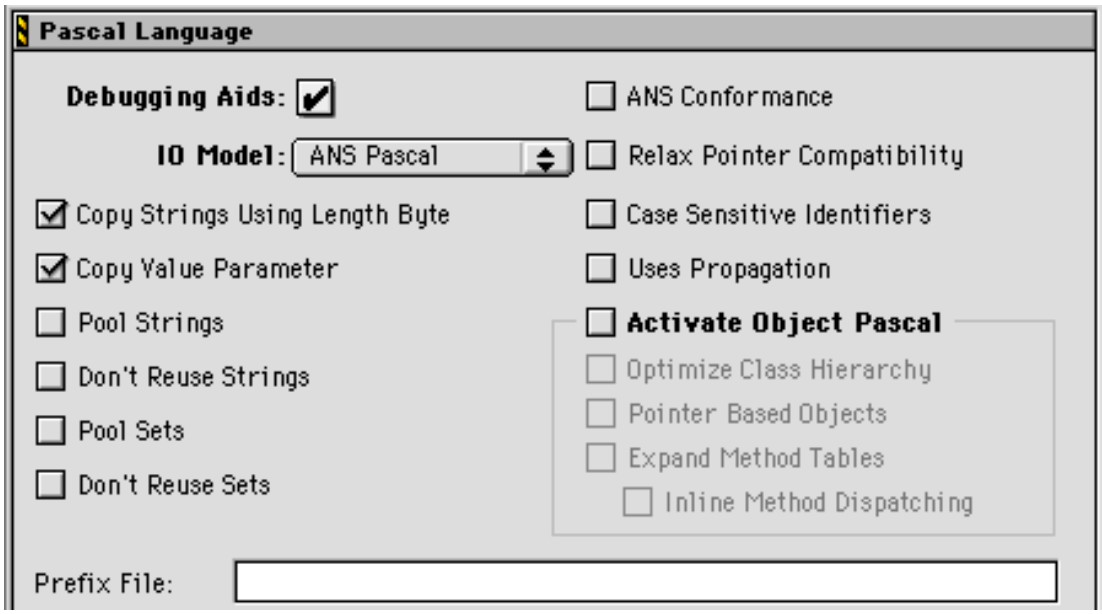


Table 11.16 Pascal Language Options Compared

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Debugging Aids	Activate Range Checking	default, -r suppresses range-checking code
Debugging Aids	Activate Overflow Checking	-ov
Debugging Aids	Trap Unmatched Case	-trapcase
Debugging Aids	Activate Nil Checking	-nilchk
IO Model	ANS Pascal	-ans
IO Model	Turbo Pascal	-turbo
IO Model	MPW Pascal	(This is the default)
Uses Propagation		(no equivalent yet)
Copy Strings Using Length Byte		-str[ings] dynamic
Copy Value Parameter		-param
Pool strings		-str[ings] pool
Don't reuse strings		-str[ings] dontreuse
Pool sets		-sets pool
Don't reuse sets		-sets dontreuse
Relax Pointer Compatibility		-relax
Case sensitive identifiers		-cs
Activate Object Pascal		(the command-line tools always use Object Pascal)
Prefix File		-prefix *filename*

Pascal Warnings

Figure 11.31 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a 68K or PPC project. Clicking **Pascal Warnings** causes options to appear in the settings panel as shown in Figure 11.32.

Table 11.16 shows the various options for the command line tools compared with those of the CodeWarrior IDE.

Figure 11.31 IDE Settings Panel (Pascal Warnings)

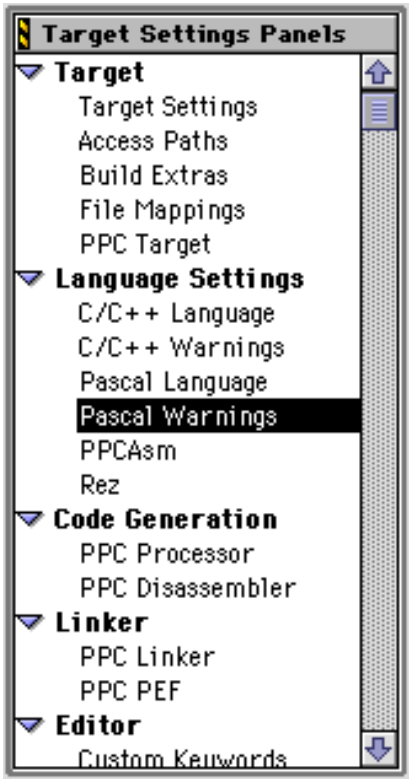


Figure 11.32 Pascal Warnings

Table 11.17 Pascal Language Options Compared

IDE Option Name	Command Line Option(s)
Modified For-Loop Indexes	-[no]foridx
Function Returns	-[no]returns
Unused Variables	-[no]unusedvar
Unused Arguments	-[no]unusedarg
Undefined Routines	-[no]routines
Check String Parameter Sizes	-[no]strings
Goto & Labels	-[no]gotos
Branching Into With	(no equivalent yet)
Branching Into For	(no equivalent yet)
Branching Between Case Legs	(no equivalent yet)
Branching Between If & Else	(no equivalent yet)

PPCAsm

Figure 11.33 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a PPC project. Selecting **PPCAsm** causes options to appear in the settings panel as shown in Figure 11.34. Additional options are shown in Figure 11.35.

Table 11.18 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.33 IDE Settings Panel (PPCAsm)

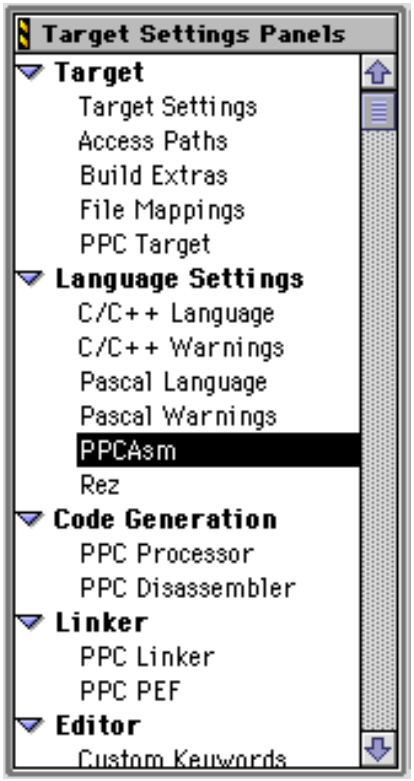


Figure 11.34 PPCAsm IDE Settings Panel

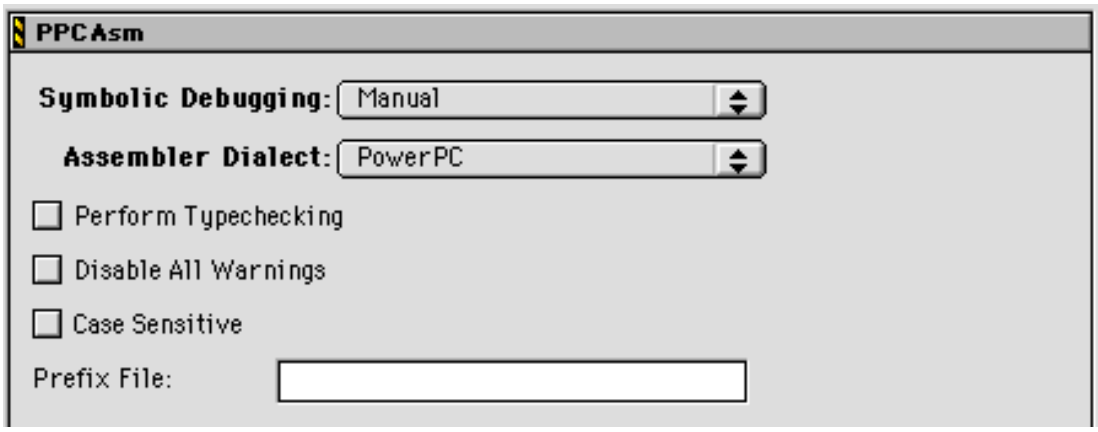


Figure 11.35 Symbolic Debugging and Assembler Dialect Popup Menus

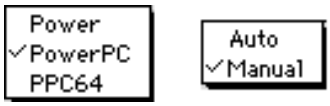


Table 11.18 PPCAsm Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Symbolic Debugging	Auto	-sym on
Symbolic Debugging	Manual	-sym man
Assembler Dialect	Power	-dialect Power
Assembler Dialect	PowerPC	-dialect PowerPC
Assembler Dialect	PPC64	-dialect PPC64
Perform typechecking		-typecheck
Disable all warnings		-w off
Case sensitive		-case on or off Default is off
Prefix File		(no equivalent for this yet)

Rez (and DeRez)

The Rez resource compiler is used for Mac OS programming to compile text resource descriptions (generally in `.r` files) into a binary format for linking with the rest of the program. The DeRez tool generally operates on `.rsrc` files, and allows you to disassemble binary resource files that have been compiled with Rez. Figure 11.36 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu. Selecting **Rez** causes options to appear in the settings panel as shown in Figure 11.37. Additional options are shown in Figure 11.38. Table 11.19 summarizes the options and their IDE equivalents.

Figure 11.36 IDE Settings Panel (Rez)

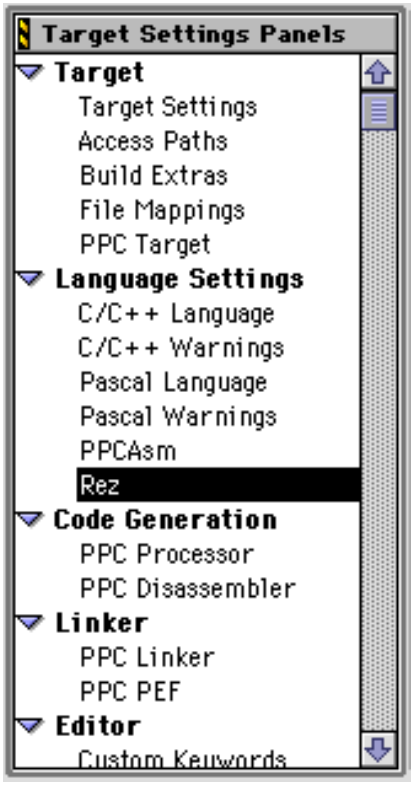


Figure 11.37 Rez Resource Compiler Options Panel

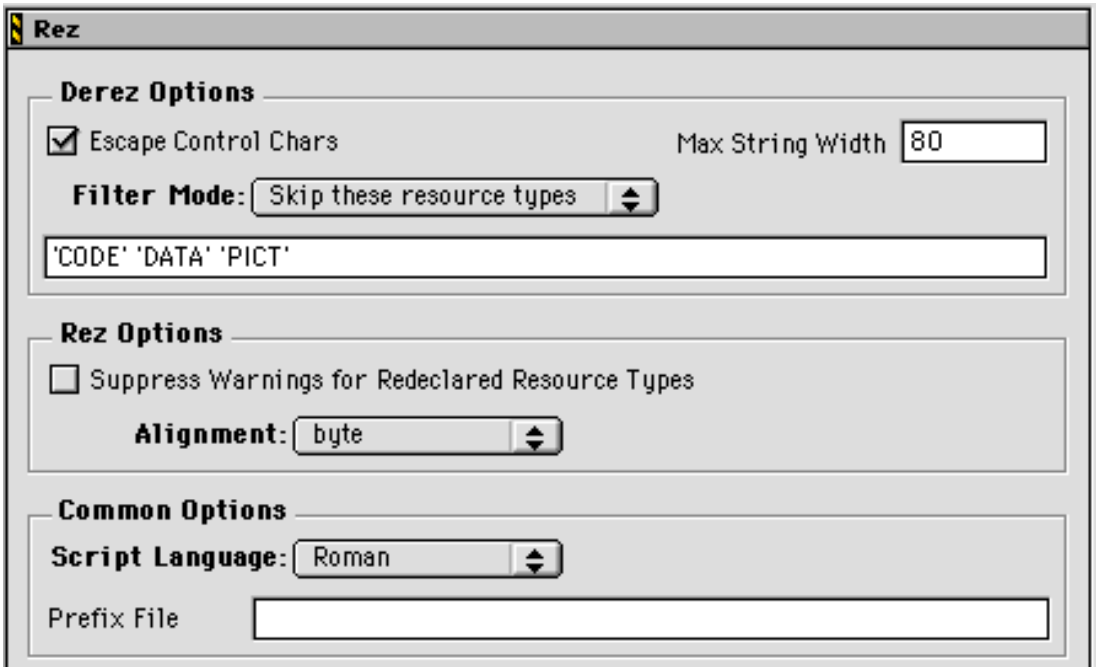


Figure 11.38 Additional Rez Pop-up Options

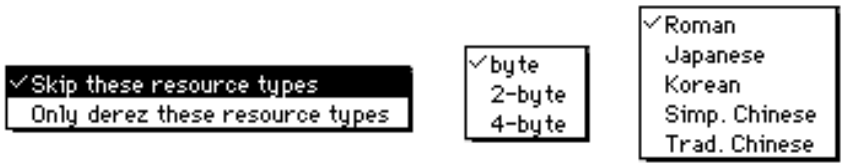


Table 11.19 Rez and DeRez Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Escape Control Chars		-e[scape] (DeRez only)
Max String Width		-m[axstringsize] n, with n between 2 and 120 (DeRez only)

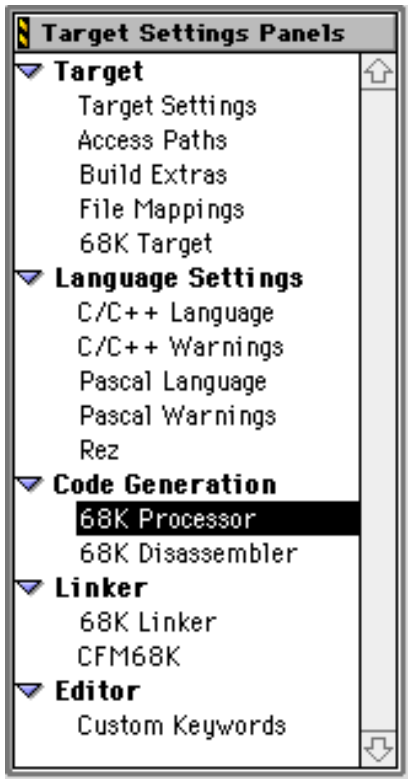
IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Filter Mode	Skip these resource types OR Only derez these resource types	-s[kip] 'typeExpr' (n:m) (DeRez only) Examples: -s "`'MENU' (1:128)`" -skip "`'MENU' (∂"MyMenu∂")`" -skip CODE
Suppress warnings for redeclared resource types		-rd (Rez only)
Alignment	byte	(no equivalent for this yet)
Alignment	2-byte	-align word (Rez only)
Alignment	4-byte	-align longword (Rez only)
Script Language		-script japanese or tradChinese or simpChinese or korean
Prefix file		Specify this option as a file on the command line. Example: `Rez MyPrefixFile.r MyRsrcs.r`

Code Generation

The CodeWarrior IDE allows customization of code generation settings for the several different tasks.

- 68K Processor Code Generation
- PPC Processor Code Generation
- 68K Disassembler
- PPC Disassembler

Figure 11.39 IDE Settings Panel (68K Processor)



68K Processor Code Generation

Figure 11.39 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a 68K project. Selecting **68K Processor** causes options to appear in the settings panel as shown in Figure 11.40. Additional options are shown in Figure 11.41.

Table 11.20 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.40 68K Processor Code Generation

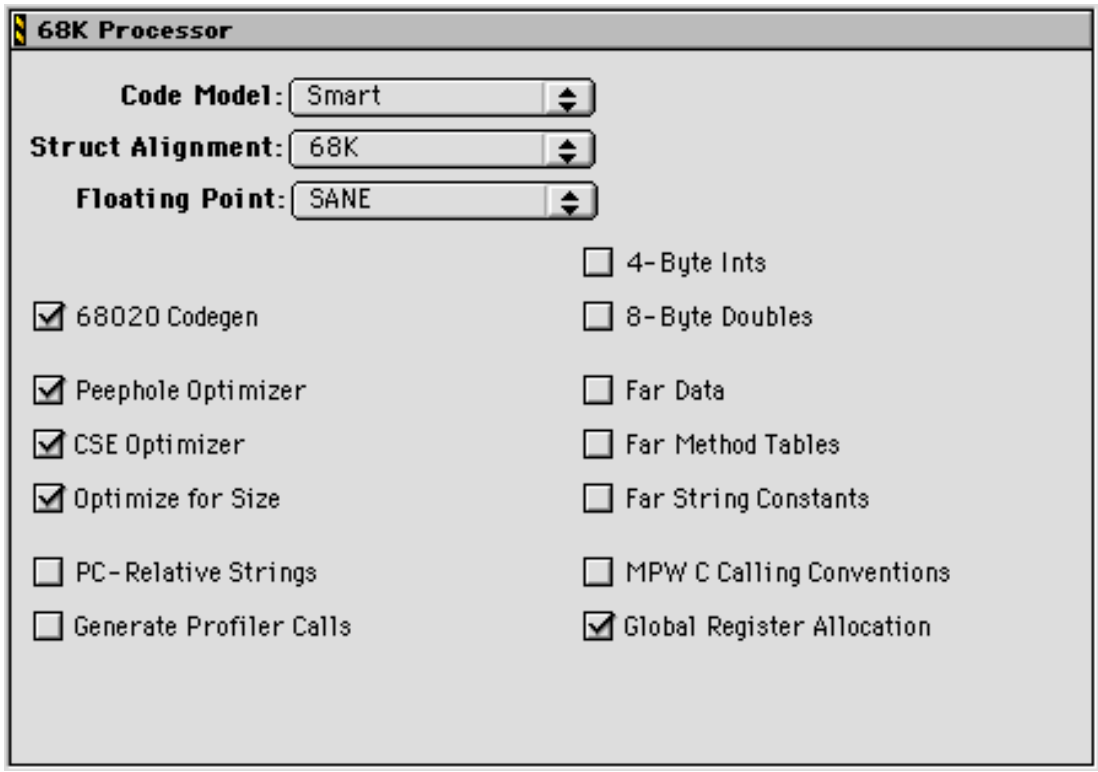


Figure 11.41 Code Model and Struct Alignment Popup Menus

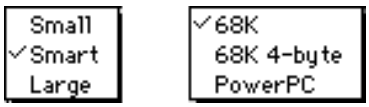


Table 11.20 68K Code Generation: MWC68K and MWPascal68K

Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Code Model	Small	-model near (Default) -model near may also be required for MWLink68K
Code Model	Smart	-model codesmart
Code Model	Large	-model far -model far may also be required for MWLink68K
Struct Alignment	68K	-align mac68k (MWC68K only)
Struct Alignment	68K 4-byte	-align mac68k4byte (MWC68K only)
Struct Alignment	PowerPC	-align power (MWC68K only)
Floating Point	SANE	(This is the default)
Floating Point	68881	-mc68881 (Mac OS) (Default is to generate SANE calls instead of code for the Motorola 68881). Required for MWC68K only.
Floating Point	Library	-fp_library on
Floating Point	PalmOS	-fp_pilot_traps on
68020 Codegen		-mc68020 (Default is to generate 68000 code).
Peephole Optimizer		-opt [no]peep
CSE Optimizer		-opt [no]cse (MWC68K only)
Optimize For Size		-opt size (MWC68K only) (Default is to optimize for speed)

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
PC-Relative Strings		-b (MWC68K only)
Generate Profiler Calls		-profile on or off (MWC68K only) (Default is off)
4-Byte Ints		-intsize 2 or 4 (MWC68K only) (4 is the default)
8-Byte Doubles		-dsize 64 or 80 (MWC68K only) (Default is 64)
Far Data		-model farData
Far Method Tables		-model farvtables or nearvtables (MWC68K only)
Far String Constants		-model farstrings or nearstrings (MWC68K only)
MPW C Calling Conventions		-mpwc (MWC68K only)
Global Register Allocation		-opt [no]color (MWC68K only) Default is on.

PPC Processor Code Generation

Figure 11.42 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a PPC project. Selecting **PPC Processor** causes options to appear in the settings panel as shown in Figure 11.43. Additional options for this settings panel are shown in Figure 11.44, as well as Figure 11.45.

Table 11.21 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.42 IDE Settings Panel (PPC Processor)

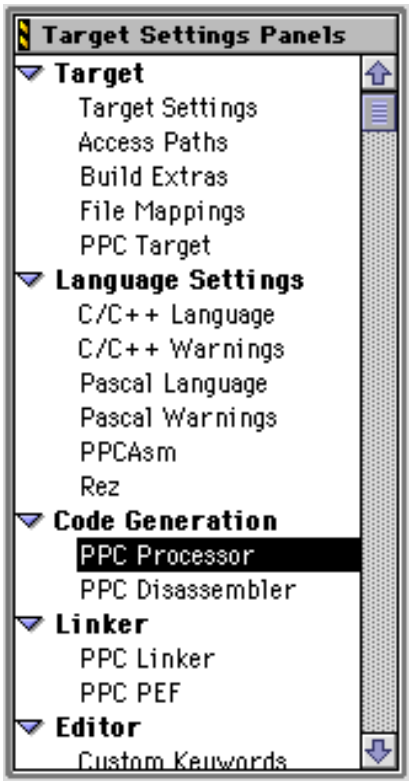


Figure 11.43 PPC Processor Code Generation

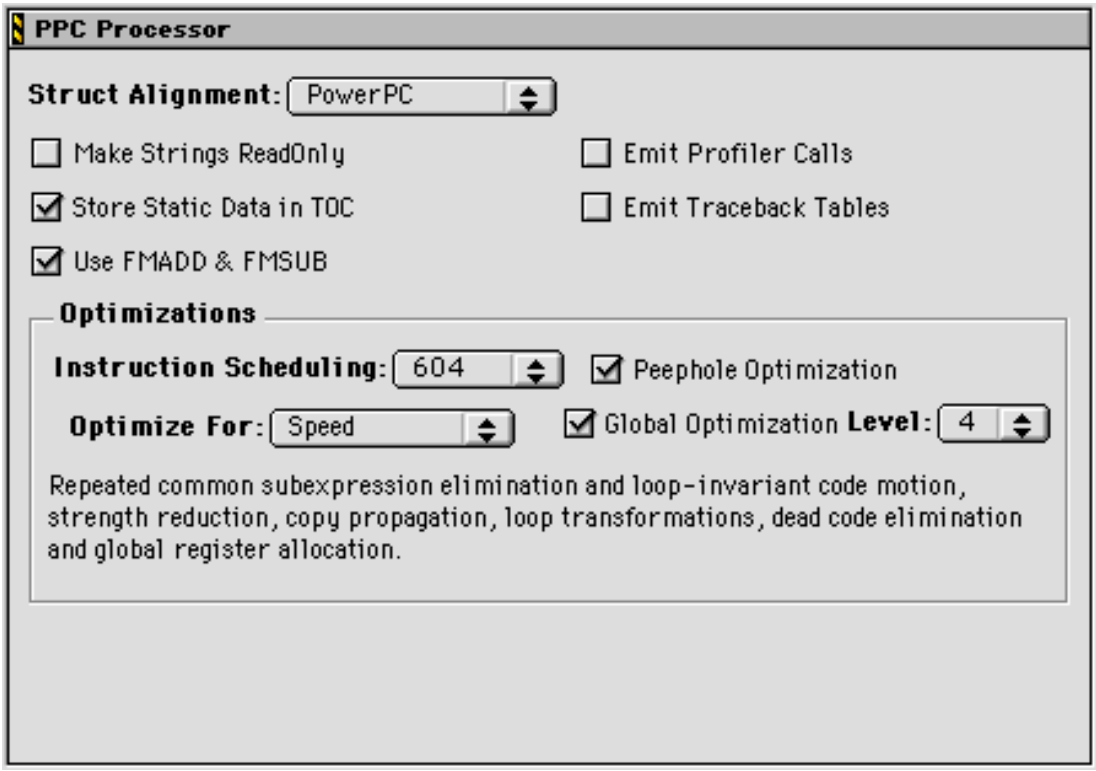

PPC Processor
Struct Alignment: PowerPC
Make Strings ReadOnly
Store Static Data in TOC
Use FMADD & FMSUB
Emit Profiler Calls
Emit Traceback Tables
Optimizations
Instruction Scheduling: 604
Peephole Optimization
Optimize For: Speed
Global Optimization Level: 4
Repeated common subexpression elimination and loop-invariant code motion, strength reduction, copy propagation, loop transformations, dead code elimination and global register allocation.

Figure 11.44 PPC Processor Code Generation Popup Menus

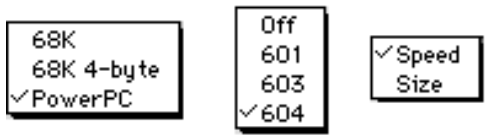


Figure 11.45 Global Optimization Level: Popup Menu

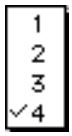


Table 11.21 PPC Code Generation: MWCPPC and MWPascalPPC (Mac OS) and mwcc/mwld (BeOS) Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Struct Alignment	68K	-align mac68k (Mac OS MWCPPC only)
Struct Alignment	68K 4-byte	-align mac68k4byte (Mac OS MWCPPC only)
Struct Alignment	PowerPC	-align power(Mac OS MWCPPC and BeOS) This option is the default
Make Strings ReadOnly		-rostr (Mac OS MWCPPC and BeOS) (Default is off)
Store Static Data in TOC		-toc_data on or off (Mac OS MWCPPC and BeOS) (Default is on)
Use FMADD and FMSUB		-fp_contract on or off (Mac OS MWCPPC and BeOS) (Default is on)
Emit Profiler Calls		-profile on or off (Mac OS MWCPPC and BeOS) (Default is off)
Emit Traceback Tables		-traceback (Mac OS MWCPPC and BeOS) (Default is no traceback tables)
Instruction Scheduling	Off	-opt noschedule (Mac OS MWCPPC and BeOS)
Instruction Scheduling	601	-opt schedule601 (Mac OS MWCPPC and BeOS) This is the default when optimization is on. Optimization is on by default.
Instruction Scheduling	603	-opt schedule603 (Mac OS MWCPPC and BeOS)

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Instruction Scheduling	604	-opt schedule604 (Mac OS MWCPPC and BeOS)
Optimize For:	Speed	-opt speed (Mac OS MWCPPC and BeOS) (Speed is the default)
Optimize For:	Size	-opt size (Mac OS MWCPPC and BeOS)
Peephole Optimization		-opt [no]peep (Mac OS and BeOS)
Global Optimization		-opt [no]global (Mac OS MWCPPC and BeOS)
Level:	1, 2, 3 or 4	-opt l1, l2, l3 or l4 (Mac OS MWCPPC and BeOS)

68K Disassembler

Figure 11.46 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a 68K project. Selecting **68K Disassembler** causes options to appear in the settings panel as shown in Figure 11.47.

Table 11.22 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.46 IDE Settings Panel (68K Disassembler)

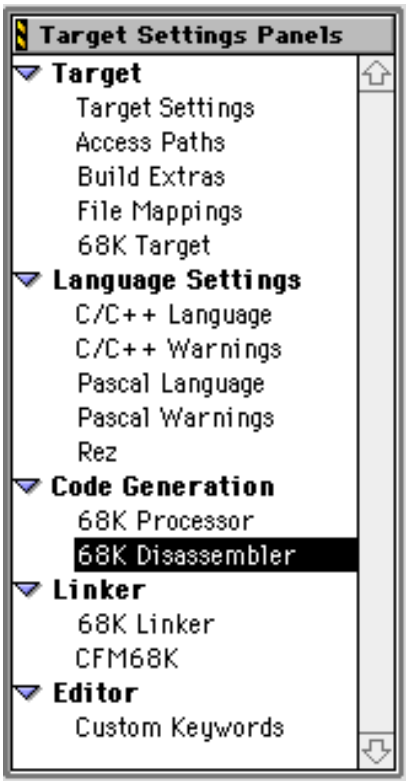


Figure 11.47 68K Disassembler Code Generation

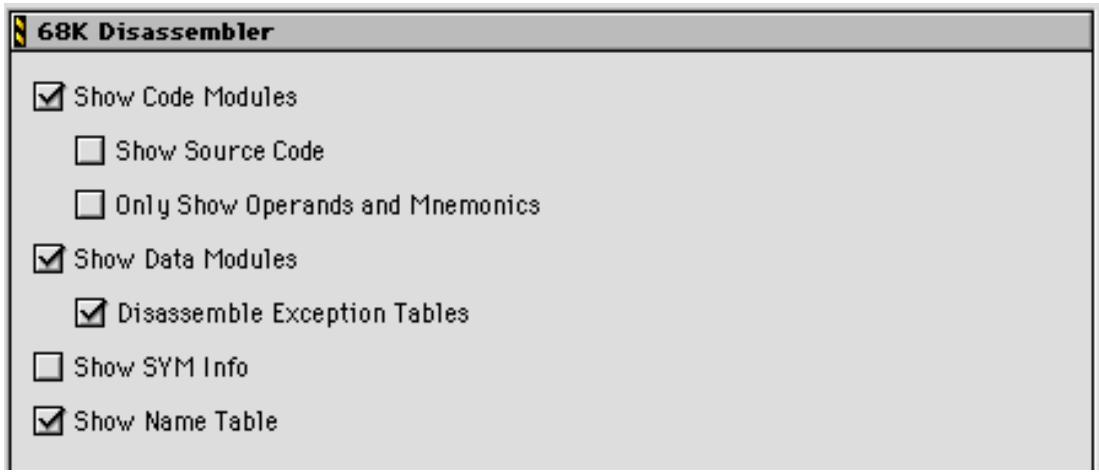


Table 11.22 68K Disassembler: MWDump68K Command Line Options

IDE Option Name	Command Line Option(s) to Enable this Behavior	Command-Line Option(s) to Disable this Behavior
Show Code Modules	(none; Show Code Modules is the default)	-d
Show Source Code	-mix	(none; off is the default)
Only Show Operands and Mnemonics	-h	(none; off is the default)
Show Data Modules	(none; Show Data Modules is the default)	-nodata
Disassemble Exception Tables	-xtables on (Default is on)	-xtables off
Show SYM Info	-sym on (Default is off)	-sym off
Show Name Table	(none; Show Name Table is the default)	-nonames

PPC Disassembler

Figure 11.48 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a PPC project. Selecting **PPC Disassembler** causes options to appear in the settings panel as shown in Figure 11.49.

Table 11.23 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.48 IDE Settings Panel (PPC Disassembler)

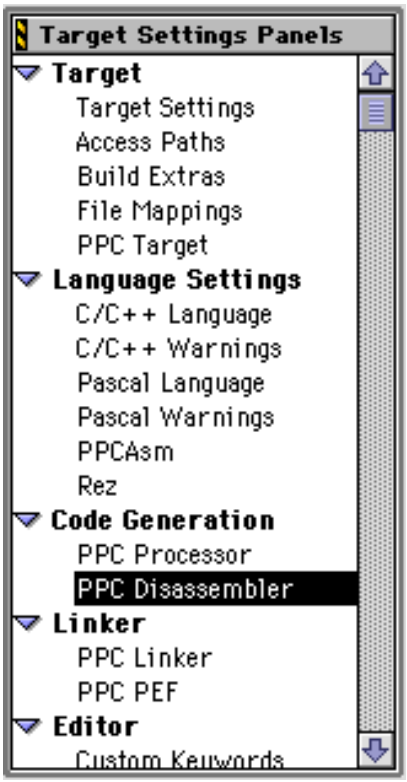


Figure 11.49 PPC Disassembler Code Generation

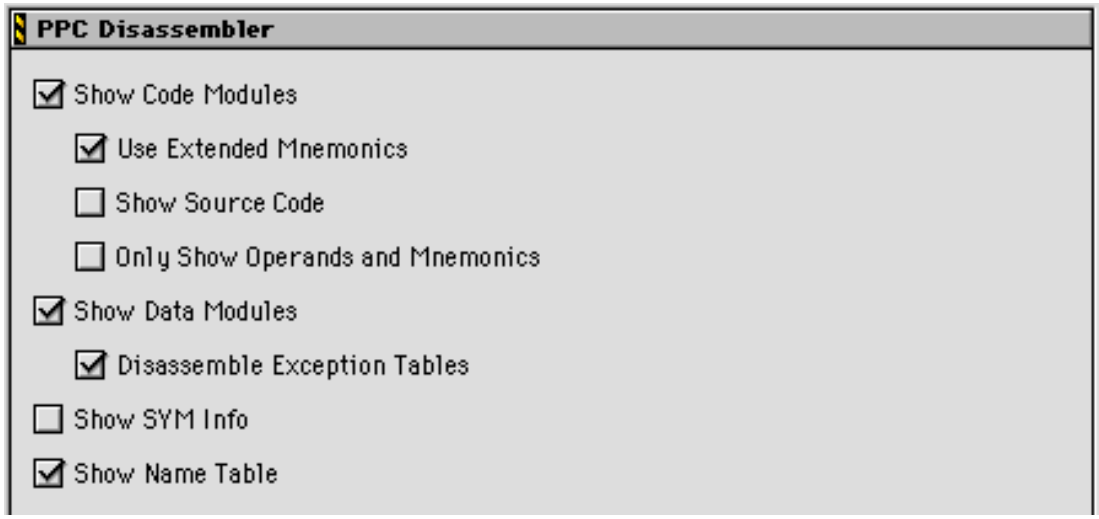


Table 11.23 PPC Disassembler: MWDumpPPC Command Line Options

IDE Option Name	Command Line Option(s) to Enable this Behavior	Command Line Option(s) to Disable this Behavior
Show Code Modules	(none; Show Code Modules is the default)	-d
Use Extended Mnemonics	-format x (The default is on)	-format nox
Show Source Code	-mix	(none; off is the default)
Only Show Operands and Mnemonics	-h	(none; off is the default)
Show Data Modules	(none; Show Data Modules is the default)	-nodata
Disassemble Exception Tables	-xtables on (Default is on)	-xtables off

IDE Option Name	Command Line Option(s) to Enable this Behavior	Command Line Option(s) to Disable this Behavior
Show SYM Info	-sym on (Default is off)	-sym off
Show Name Table	(none; Show Name Table is the default)	-nonames

Linker

The CodeWarrior IDE allows customization of several different types of linker settings.

- 68K Linker
- CFM68K Linker
- PPC Linker
- PPC PEF

68K Linker

Figure 11.50 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a 68K project. Selecting **68K Linker** causes options to appear in the settings panel as shown in Figure 11.51. Additional options for this settings panel are shown in Figure 11.52.

Table 11.24 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.50 IDE Settings Panel (68K Linker)

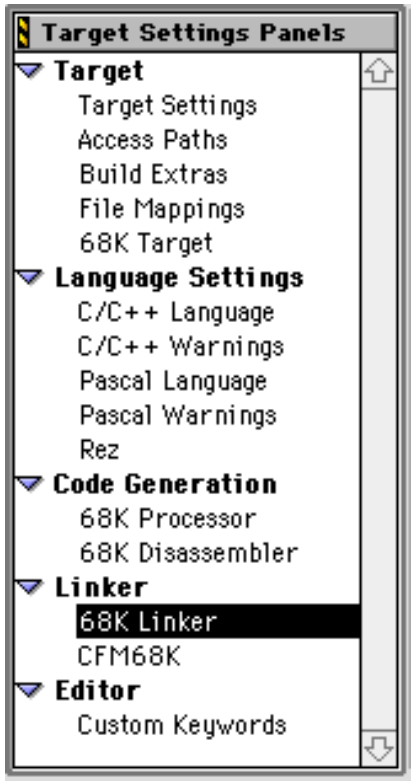


Figure 11.51 68K Linker Settings Panel

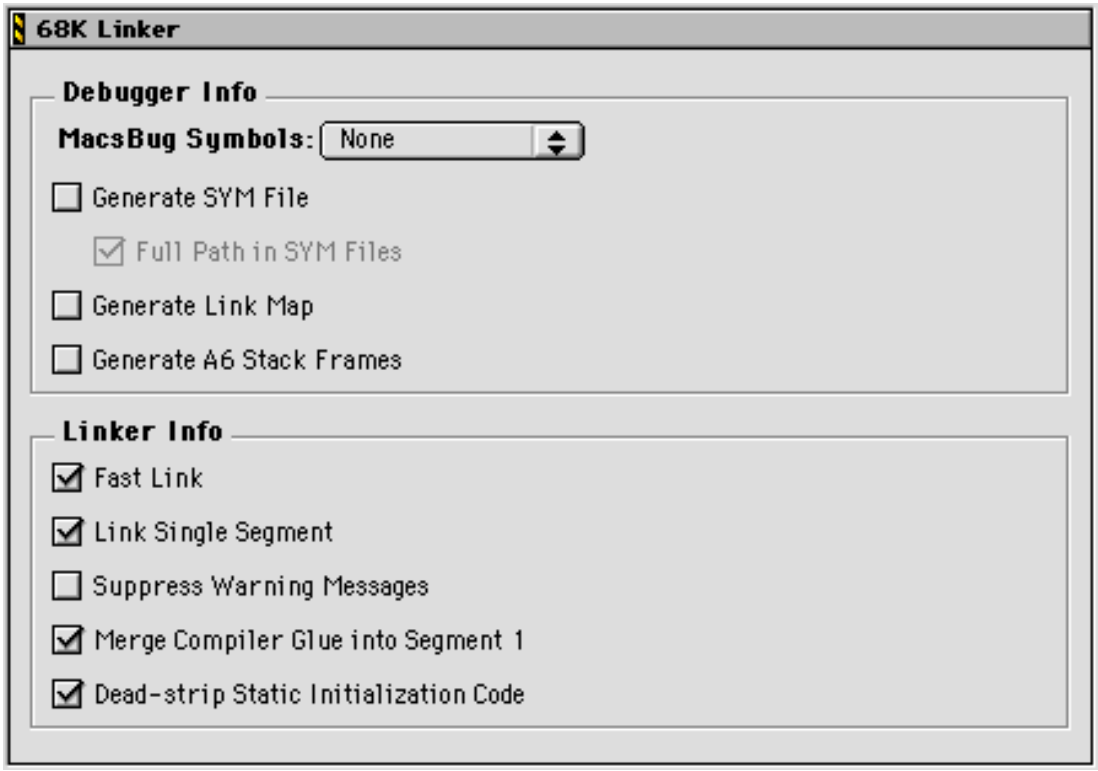


Figure 11.52 MacsBug Symbols Popup Menu

Table 11.24 68K Linker: MWLink68K/MWC68K/MWPascal68K

Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option	Command Line Tool(s)
MacsBug Symbols	None	-mbg off	MWC68K, MWPascal68K
MacsBug Symbols	Old Style	-mbg ch8	MWC68K, MWPascal68K
MacsBug Symbols	New Style	-mbg on or full (The full option turns on A6 stack frames as does the -a6 option)	MWC68K, MWPascal68K
Generate SYM File		-sym on (Mac OS) The SYM file creator can be set with -csym type (default for type is 'MWDB'). -sym 3.2	MWC68K, MWPascal68K, MWLink68K
Full Path in SYM Files		-sym fullpath (Mac OS) (use -sym full for Jasik The Debugger support)	MWC68K, MWLink68K
Generate Link Map		-map filename (Mac OS) The mapfile creator can be set with -cmap type, where 'MPS ' is the default. Use 'CWIE' for making the link map double-clickable for the Codewarrior debugger.	MWLink68K
Generate A6 Stack Frames		-[no]a6	MWC68K
Fast Link		-fastlink on or off (Default is on for Mac OS) Note: Required to be on for #pragma segment support	MWLink68K

IDE Option Name	IDE Choice (if applicable)	Command Line Option	Command Line Tool(s)
Link Single Segment		-single	MWLink68K
Suppress Warning Messages		-w[arn]	MWLink68K
Merge Compiler Glue into Segment 1		-glue mainseg or curseg (default is mainseg)	MWLink68K
Dead-Strip Static Initialization Code		-dead initcode (Mac OS) -dead off	MWLink68K

CFM68K Linker

Figure 11.53 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a 68K project. Selecting **CFM68K** causes options to appear in the settings panel as shown in Figure 11.54.

Table 11.25 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.53 IDE Settings Panel (CFM68K Linker)

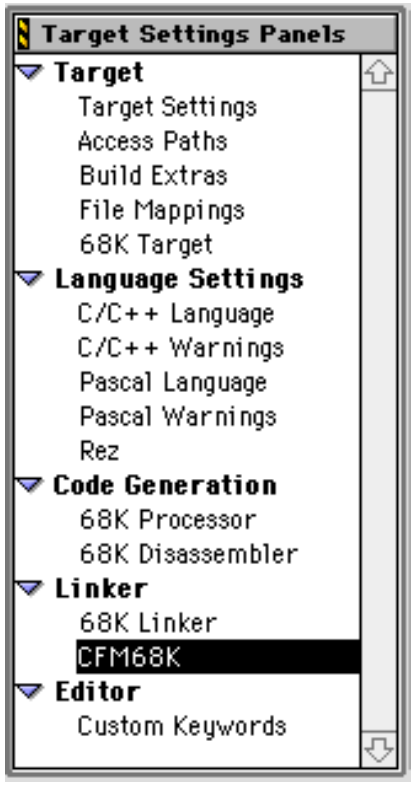


Figure 11.54 CFM68K Settings Panel

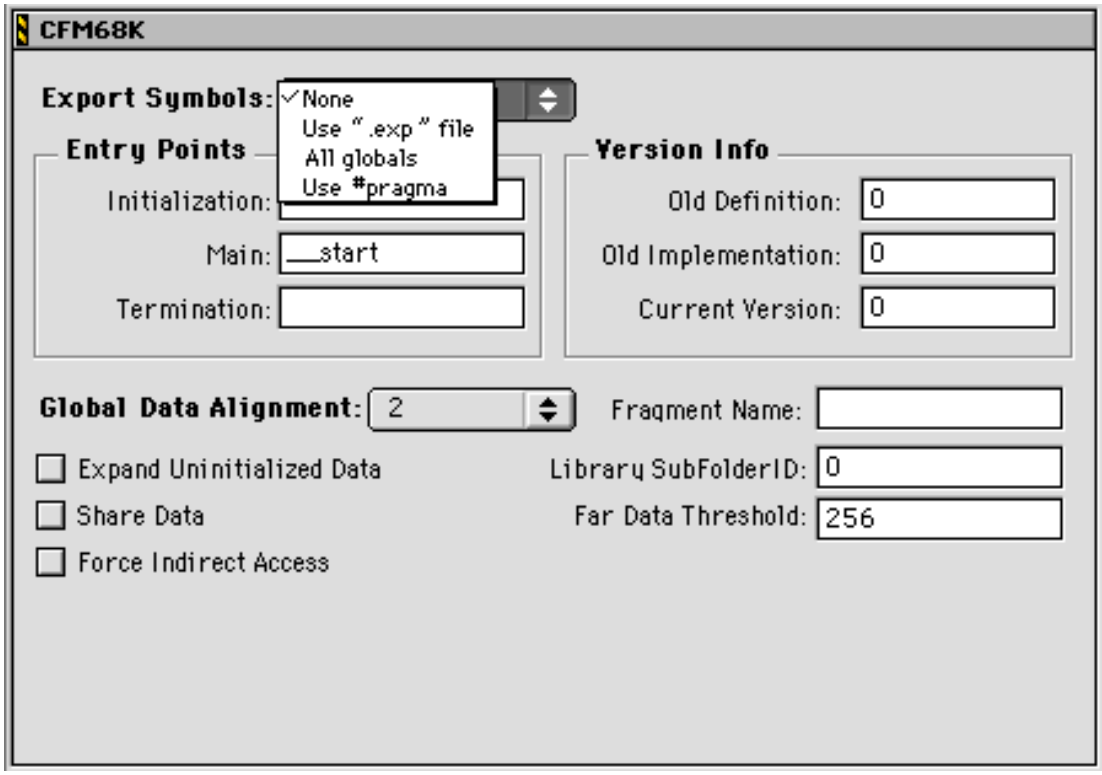


Table 11.25 CFM68K Linker: MWLink68K/MWC68K Command Line

Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Export Symbols	None	-export none (MWLink68K only) (none is the default for -export)
Export Symbols	Use ".exp" file	-@export filename (MWLink68K only)
Export Symbols	All globals	-export all (MWLink68K only)
Export Symbols	Use #pragma	-export pragma (MWLink68K only)
Old Definition		-dv num (MWLink68K only) Default for num is 0
Old Implementation		-uv num (MWLink68K only) Default for num is 0
Current Version		-cv num (MWLink68K only) Default for num is 0
Initialization		-init name (MWLink68K only) Default is no init routine
Main		-m[ain] name (MWLink68K only) Default is __start
Termination		-term name (MWLink68K only) Default is no termination routine
Global Data Alignment	1, 2, 4, or 8	-ad *number*
Share Data		-sharedata (MWLink68K only)
Expand Uninitialized Data		-b (MWLink68K only)
Force Indirect Access		-indirect (MWC68K tool only)
Fragment Name		-name pefname (MWLink68K only) Default for pefname is the output file name

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Library SubFolder ID		-appSub[FolderID] *IDNum* Default is kNoAppSubFolder (MWLink68K only)
Far Data Threshold		-fdt num (MWC68K tool only) Default for num is 256

PPC Linker

Figure 11.55 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when working with a PPC project. Selecting **PPC Linker** causes options to appear in the settings panel as shown in Figure 11.56. Additional options for this settings panel are shown in Figure 11.57.

Table 11.26 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.55 IDE Settings Panel (PPC Linker)

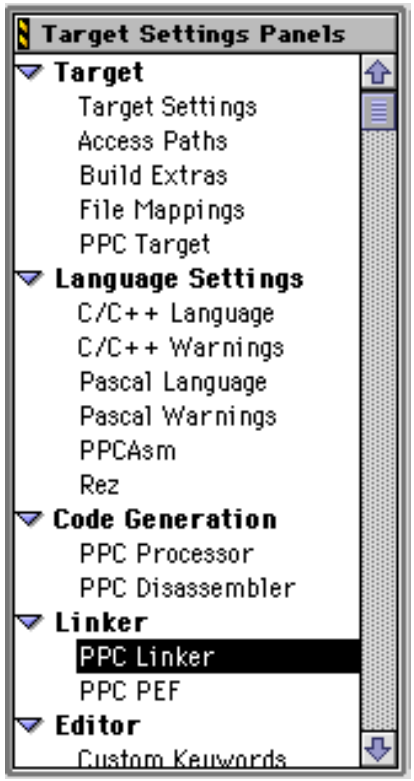


Figure 11.56 PPC Linker: MWLinkPPC Command Line Options

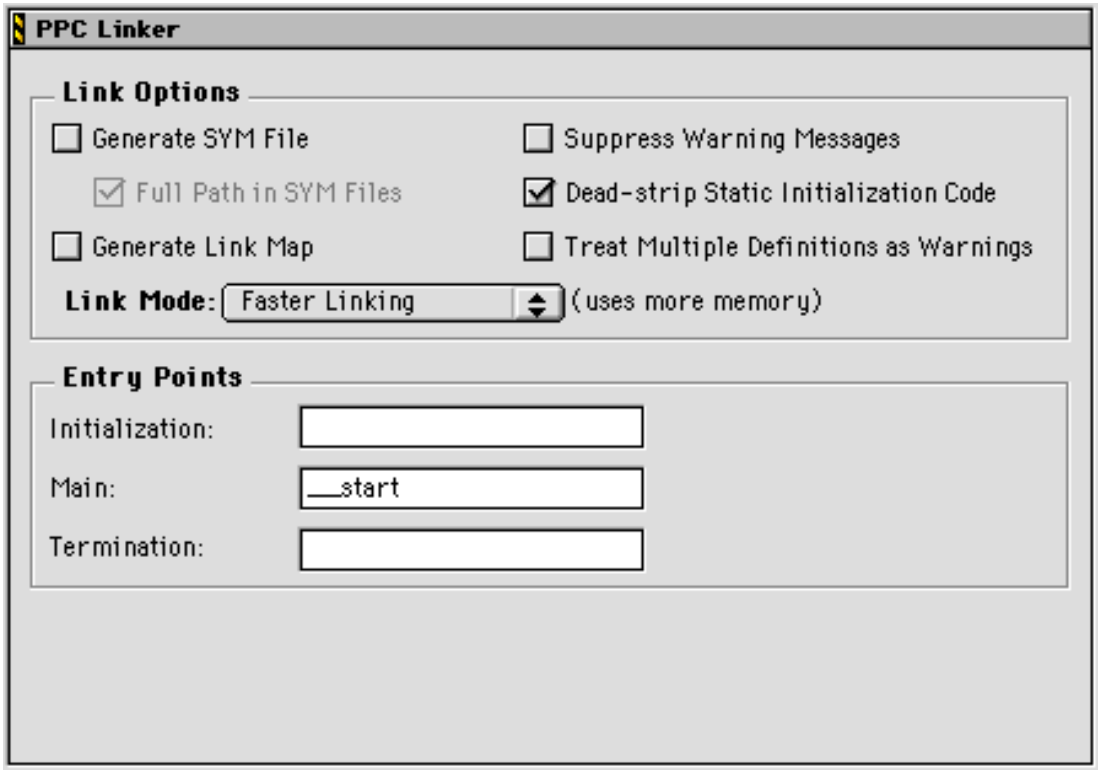


Figure 11.57 PPC Linker Speed Popup Menu

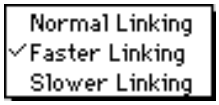


Table 11.26 PPC Linker: MWLinkPPC (Mac OS) and mwcc/mwld

(BeOS) Command Line Options

IDE Option Name	Command Line Option(s)
Generate SYM File	-sym on (Mac OS and BeOS) The SYM file creator can be set with -csym type. Default for type is 'MWDB'. (Mac OS only)
Use Full Path Names	-sym full (Mac OS and BeOS)
Generate Link Map	-map filename (Mac OS and BeOS)
Suppress Warning Messages	-w[arn] (Mac OS and BeOS)
Dead-strip Static Initialization Code	-dead initcode (Mac OS and BeOS)
Treat Multiple Definitions as Warnings	-multidefs (Mac OS only and BeOS) Default is to treat as errors instead of warnings
Normal Linking	-link normal (Mac OS and BeOS)
Faster Linking	-link fast[er] (Mac OS and BeOS)
Slower Linking	-link slow[er] (Mac OS only)
Initialization	-init name (Mac OS and BeOS) Default is no init routine (Mac OS) Default is _init_routine_ (BeOS)
Main	-m[ain] name (Mac OS and BeOS) Default is __start (Mac OS and BeOS) If using BeOS and making a shared library this should not be specified
Termination	-term name (Mac OS and BeOS) Default is no termination routine (Mac OS) Default is _term_routine_ (BeOS)

PPC PEF

Figure 11.58 shows the settings panel. This panel appears when you choose the **Target Settings** menu item in the **Edit** menu, when

working with a PPC project. Selecting **PPC PEF** causes options to appear in the settings panel as shown in Figure 11.59.

Table 11.27 shows the command line tool options as compared to the CodeWarrior IDE options.

Figure 11.58 IDE Settings Panel (PPC PEF)

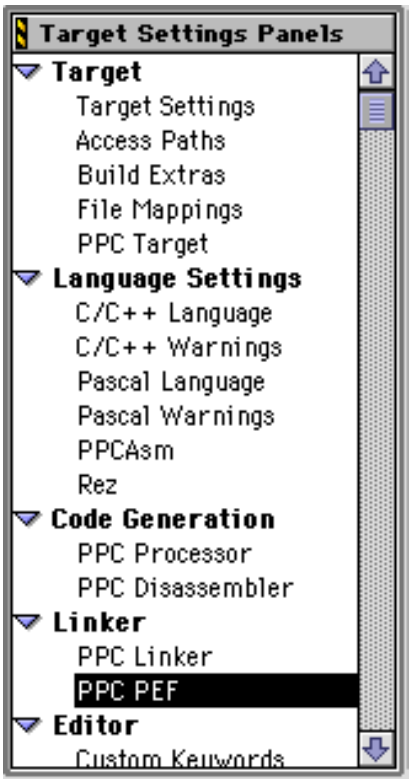


Figure 11.59 IDE Settings Panel (PPC PEF)

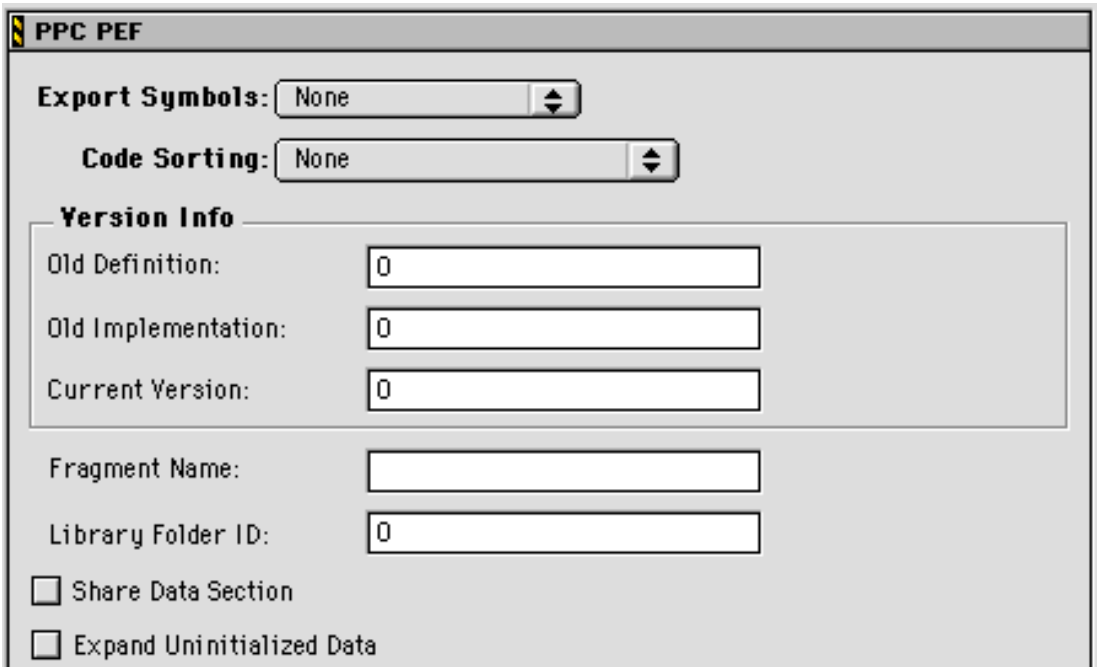


Figure 11.60 PowerPC PEF Export Symbols and Code Sorting Pop-ups

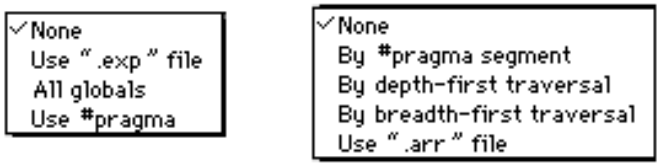


Table 11.27 PPC Linker: MWLinkPPC Command Line Options

IDE Option Name	IDE Choice (if applicable)	Command Line Option(s)
Export Symbols	None	-export none (none is the default for -export)
Export Symbols	Use ".exp" file	-@export <expfile>.exp
Export Symbols	All globals	-export all
Export Symbols	Use #pragma	-export pragma
Code Sorting	None	-codeorder none (this is the default)
Code Sorting	By #pragma segment	-codeorder segment
Code Sorting	By depth-first traversal	-codeorder depthfirst
Code Sorting	By breadth-first traversal	-codeorder breadthfirst
Code Sorting	Use ".arr" file	-codeorder filename (checks file filename.arr)
Code Sorting	By #pragma segment	-sortcode
Old Definition		-dv num Default for num is 0
Old Implementation		-uv num Default for num is 0
Current Version		-cv num Default for num is 0
Fragment Name		-name pefname Default is output file name
Library Folder ID		-appSub[FolderID] id Default is kNoAppSubFolder
Share Data Section		-sharedata
Expand Uninitialized Data		-b

12

Pragmas and Command-Line Options

This chapter describes the correspondence between `#pragma` directives in source files and the Metrowerks command-line tools.

Overview of Pragmas and Options

The source code files in a project may have `#pragma` directives embedded in them to indicate special processing needs. These pragmas are evaluated by the Metrowerks command-line tools and the appropriate actions taken to produce the correct object file output. This chapter maps the available pragmas to the command-line tool options that produce the equivalent effect. Table 12.1 shows the relationships between pragmas and tools.

For example, a file `foobar.cpp` may have the `#pragma ARM_conform on` directive embedded somewhere in the file, indicating that the file should be compiled with ARM conformance. The command-line option for this (passed to a command-line tool like MWC68K) would be -ARM on.

For the MWC68K and MWCPPC (with BeOS) command-line tools, any pragma that does not have a command-line option can be passed to the compiler with the -pragma *pragmaName* syntax. For example:

```
-pragma "export on"
```

In this case, the pragma is inserted at the top of the file that the -pragma option is passed to.

A verbose explanation of the available `#pragma` directives in the Metrowerks command-line tools can be found in the *C Compiler Guide* manual.

BeOS supports all options listed here unless specified otherwise.

Table 12.1 Pragmas and Equivalent Command-Line Options

#pragma	Command-line option (if available)
a6frames	-a6 (only for MWC68K tool and Mac OS)
align	-align
align_array_members	-align array (Mac OS only)
ANSI_strict	-strict on
ARM_conform	-ARM
auto_inline	-inline auto
bool	-bool
check_header_flags	-check precomp
code_seg	(ignored Win32/x86 only)
code68020	-mc68020 (Mac OS 68K only)
code68881	-mc68881 (Mac OS 68K only)
cplusplus	-dialect Cplus
cpp_extensions	-msext
d0_pointers	-d0 (Mac OS 68K only)
data_seg	(ignored Win32/x86 only)
direct_destruction	(compilers now ignore this pragma)
direct_to_som	-som
disable_registers	(Mac OS PowerPC only)
dont_inline	-inline off

#pragma	Command-line option (if available)
dont_reuse_strings	-str[ings] dontreuse
enumsalwaysints	-enum int or -enum min
exceptions	-Cpp_exceptions
export	-export
extended_errorcheck	-w[arnings] [no]extended
far_code	-model farCode (Mac OS 68K only)
far_data	-model farData (Mac OS 68K only)
far_strings	-model farstrings (Mac OS 68K only)
far_vtables	-mode farVtables (Mac OS 68K only)
force_active	-force_active (Mac OS 68K only)
fourbyteints	-intsize 4 (Mac OS 68K only)
fp_contract	-fp_contract (Mac OS PowerPC and BeOS only)
fp_library	-fp_library (68K only)
fp_pilot_traps	-fp_pilot_traps (PalmOS only)
function	(ignored Win32/x86 only)
global_optimizer	-opt [no]global (Mac OS PowerPC and BeOS only)
IEEEdoubles	-dsize 64 (Mac OS 68K only)
ignore_oldstyle	-proto ignoreoldstyle
import	(Mac OS only)
inline_depth	(ignored Win32/x86 only)
internal	(Mac OS only)
lib_export	(Mac OS only)
longlong	(MacOS only)
macsbug	-mbg on or off (Mac OS 68K only)

#pragma	Command-line option (if available)
mark	
mpwc	-mpwc (Mac OS 68K only)
mpwc_newline	-mapcr
mpwc_relax	-mpw_chars
near_code	-model nearCode (Mac OS 68K only)
no_register_coloring	-[no]color (MacOS 68K only)
once	-once
oldstyle_symbols	-mbg ch8 (Mac OS 68K only)
only_std_keywords	-appleext off
optimization_level	-l1, -l2, -l3 or -l4 (Mac OS PowerPC and BeOS only)
optimize_for_size	-opt size
pack	(Win32/x86 only)
parameter	(Mac OS 68K only)
pcrelstrings	-b (Mac OS 68K only)
peephole	-opt [no]peep (Mac OS PowerPC and BeOS only)
pointers_in_A0	(this is the default)
pointers_in_D0	-d0
pool_strings	-str[ings] pool (PowerPC only)
precompile_target	(ignored, the filename is set with -precompile on the command line)
profile	-profile on or off (Mac OS only)
pop	
push	
readonly_strings	-rostr (PowerPC only)
require_prototypes	-r[equireprotos]

#pragma	Command-line option (if available)
RTTI	-RTTI on or off
scheduling	-opt [no]schedule (Mac OS PowerPC and BeOS only for these) -opt [no]schedule601 -opt [no]schedule603 -opt [no]schedule604
segment	(Mac OS 68K only) This is generally ignored on PowerPC by MWCPPC unless -sortcode is specified on MWLinkPPC command line
side_effects	(Mac OS only)
smart_code	-model codesmart (Mac OS 68K only)
static_inlines	-inline static
sym	-sym *value*
toc_data	-toc_data on or off (Mac OS PowerPC and BeOS only)
trigraphs	-trigraphs on or off
traceback	-traceback (PowerPC only)
unsigned_char	-char unsigned
unused	
warn_emptydecl	-w[arnings] [no]emptydecl
warning_errors	-w[arnings] [is]err[or]
warn_extracomma	-w[arnings] [no]extracomma
warn_hidevirtual	-w[arnings] [no]hidevirtual
warn_illpragma	-w[arnings] [no]pragmas
warn_possunwant	-w[arnings] [no]possible
warn_unusedarg	-w[arnings] [no]unusedarg
warn_unusedvar	-w[arnings] [no]unusedvar
warning	(Win32/x86 only, ignored)

Index

Symbols

-ad option 136
-align option 103
-b option 147
-c option 88
-clean option 119
-codeorder option 137
-cv option 147
-D option 76
-d option 155
-fc option 148
-ft option 148
-G option 148
-g option 158
-h option 117
-help option 153
-k option 119
-L option 149
-l option 148
-make option 92
-maxerrors option 154
-mergefragment option 143
-multidefs option 154
-nodefaults option 154
-nodirectdestruction option 68
-nodup option 157
-noload option 119
-noprejudge option 155
-O option 71
-ov option 110
-r option 110
-rebuild option 117
-s option 72
-som option 73
-sym option 98, 119
-u option 79
-version option 157
-w option 93
-xma option 150
-xml option 151
-xms option 152

A

-a4 option 56, 102, 161, 165
-a6 option 56, 103
-align option 57
-ans option 111
-ansi option 79
-appleext option 80
application segmenting 44
-appSub option 136
-ARM option 80

B

-b option 58, 104, 137
-b2 option 58, 104
-bool option 81

C

C MPW compiler
    compared to MWC68K 170
-c option 81, 111
-carr option 137
-cexp option 155
-cfrg option 137
-char option 82
-check precompflags option 75
-cmap option 155
code resources
    -rsrcmulti 47
    default name 37
    multi-segment 47
    single-segment 46
-coderesource option 131
-collapsereloads option 122
combining segments 38
CPlus MPW compiler
    compared to MWC68K 175
-Cpp_exceptions option 59
-cs option 112
-csym option 157
custom headers 51
-custom option 123, 131
-cv option 138

-cwd option 88, 115

D

-d option 75, 111, 162, 166
-d0 option 59, 104
-da option 50, 131
-dead option 123
-g option 97
default attributes 52
-dialect option 82
-directdestruction option 59
disassemble code 165
-drvr option 132
-dsize option 60, 104
-dump option 76
-dv option 138

E

-e option 76, 117, 139
-ec option 82
-enum option 61
-EOL option 83
-@export option 139
-export option 139

F

-f option 140
-fatext option 61, 105
-fdt option 61
fmt option 162, 166
-force_active option 62, 105
-format option 162, 166
-fp_contract option 62, 106
-fp_library option 62
-fp_pilot_traps option 63, 260

G

-glue option 125

H

-h option 162, 166
-help option 92

I

-i- option 89, 115
-ib option 141
-IDEseg option 41
-IDEsegment option 126
include file searching 29
-indirect option 64
-inline option 63
-wnotinlined option 96
-intsize option 63, 123

J

-j0 option 83
-j1 option 83
-j2 option 83
-jm option 92

L

-ldsize option 64, 106
libraries
    no segmentation 44
-libraryA4 option 148
-librename option 142
Link MPW linker
    compared to MWLink68K 199
-link option 153
-load option 77

M

-m option 163, 167
-ma option 126
-main option 142
-make option 64
-map option 155
-mapcr option 83
-mapIDE option 156
-maxerrors option 92, 117
-mbg option 97, 118
-mc68020 option 64, 106
-mc68881 option 65, 106
-merge option 132
-mf option 77, 112, 154

-mix option 167
-model option 65, 66, 67, 107, 108, 143
MPW tools, segmenting 44
-mpw_pointers option 84
-mpwc option 65, 84
MrC/MrCpp compiler
    compared to MWCPPC compiler 192
    optimization 192
-msext option 85, 288
-msg option 156
multi-segment code resources 47
MWC68K compiler
    compared to C compiler 170
    compared to CPlus compiler 175
    compared to SC/Cpp compiler 181
    optimization 69, 109
MWCPPC compiler
    compared to MrC/MrCpp compiler 192
    compared to PPCC compiler 186
    optimization 70, 109, 187, 192
MWDump68K 165
MWDumpPPC 161, 165
MWLink68K
    compared to Link MPW linker 199
MWLinkPPC
    compared to PPCLink 203

N

-n option 163, 167
-name option 144
naming segments 37
-native option 133
-nilchk option 112
-nodefaults option 91
-nofail option 92
-nomf option 77, 112
-nostdinc option 91

O

-o option 68, 108
-objc option 84
-once option 77
-opt option 68, 108
optimizing options
    MWC68K 69, 109
    MWCPPC 70, 109, 187, 192

P

-p option 93, 117, 157
-param option 112
PPCC compiler
    compared to MWCPPC 186
    optimization 187
PPCLink
    compared to MWLinkPPC 203
#pragma segment 35, 41, 48
    and dispatch segment 51
    and libraries 44
-pragma option 77
-precompile option 78
-prefix option 78
-profile option 98, 120
-proto option 85

R

-ra option 52
-relax option 112
renaming segments 37
    multiple 39
    with no name 42
-requireprotos option 85
resource attributes
    default 52
    in code resources 53
    setting 52
-rn option 129
-rostr option 71
-rsrcflags option 54, 133
-rsrcmerge option 133
-rsrcmulti option 134
    and code resources 47
-rsrcname option 48, 134
    and code resources 46
-rsrcsegtype option 134
-rsrcvirfuncs option 135
-rt option 135
-RTTI option (run-time type information) 72, 85

S

-s option 129
-safe option 135
-sb option 42, 129
  and code resources 46
  and libraries 44
  compared to -s 41
  syntax 41
SC/Cpp compiler
  compared to MWC68K 181
searching, include files 29
segmenting
  applications 44
  libraries 44
  MPW Tools 44
  requires fast linking 36
segments
  combining 38
  renaming 37
  renaming multiple 39
  with no name 42
-sets option 113
-sf option 130
-sg option 130
  and MPW tools 44
  compared to -sn 39
  without name 39
-sharedata option 144
-single option 41, 130
  and code resource 47
  syntax 41
single-segment code resources 46
-size option 124
-sizeflags option 125
-sizemax option 124
-sizemin option 125
-sn option 130
  and code resources 49
  renaming Main segment 51
-som_env_check option 73
-sortcode option 144
-stacksize option 145
-str option 74
-strict option 86
-strings option 73, 111, 113
-sym option 97, 119, 163, 167

T

-t option 93, 117
-tdb option 159
-term option 145
-tmp option 130
-toc_data option 74
-traceback option 98
-trapcase option 113
-trigraphs option 86

U

-U option 79
-uv option 145

V

-version option 93

W

-warnings option 93, 117
-wchar_t option 86
-weaklib option 146
-wi option 147
-wimplicitconv option 96

X

-xm application option 150
-xm code resource option 136

CodeWarrior

MPW Cmd-Line Tools Reference

Credits

writing lead:	BitHead
other writers:	Stephen Jovanovic, Marc Paquette, Jim Trudeau
engineering:	Mark Anderson, Jon Watte, Marcel Achim
frontline warriors:	Mark Anderson, Eric Hunt, Chris Kueny, Ron Levreault, Dave Newman, Fred Peterson, Khurram Qureshi, Eric Clapton, and CodeWarrior users everywhere

Guide to CodeWarrior Documentation

CodeWarrior documentation is modular, like the underlying tools. There are manuals for the core tools, languages, libraries, and targets. The exact documentation provided with any CodeWarrior product is tailored to the tools included with the product. Your product will not have every manual listed here. However, you will probably have additional manuals (not listed here) for utilities or other software specific to your product.

Core Documentation	
IDE User Guide	How to use the CodeWarrior IDE
Debugger User Guide	How to use the CodeWarrior debugger
CodeWarrior Tutorials	Step-by-step introduction to IDE components
Language/Compiler Documentation	
C Compiler Guide	Information on the C/C++ compiler for any target
Pascal Compiler Guide	Information on the Pascal compiler for any target
Error Reference	Comprehensive list of compiler/linker error messages, with many fixes
Pascal Language Reference	The Metrowerks implementation of ANS Pascal
Assembler Guide	Stand-alone assembler syntax
MPW Cmd-Line Tools Reference	Command-line options for MPW
Plugin API Manual	The CodeWarrior plugin compiler/linker API
Library Documentation	
MSL C Reference	Function reference for the Metrowerks standard C library
MSL C++ Reference	Function reference for the Metrowerks standard C++ library
Pascal Library Reference	Function reference for the Metrowerks ANS Pascal library
The PowerPlant Book	Introductory guide to the Metrowerks application framework for Mac OS
PowerPlant Advanced Topics	Advanced topics in PowerPlant programming for Mac OS
MFC Reference	Reference for the Microsoft Foundation Classes for Win32
Win32 SDK Reference	Microsoft's Reference for the Win32 API
Targeting Manuals	
Targeting BeOS	How to use CodeWarrior to program for BeOS
Targeting Java VM	How to use CodeWarrior to program for the Java Virtual Machine
Targeting Mac OS	How to use CodeWarrior to program for Mac OS
Targeting MIPS	How to use CodeWarrior to program for MIPS embedded processors
Targeting Palm OS	How to use CodeWarrior to program for PalmPilot
Targeting PlayStation	How to use CodeWarrior to program for the PlayStation game console
Targeting PowerPC Embedded Systems	How to use CodeWarrior to program for PPC embedded processors
Targeting Win32	How to use CodeWarrior to program for Windows 95/NT